Proceedings of the 3rd Workshop on

New Frontiers in Mining Complex Patterns (NFMCP 2014)

Editors

Annalisa Appice

Michelangelo Ceci

Corrado Loglisci

Giuseppe Manco

Elio Masciari

Zbigniew W. Ras

New Frontiers in Mining Complex Patterns (NFMCP 2014)

The third International Workshop on New Frontiers in Mining Complex Patterns (NFMCP 2014) was held in Nancy in conjunction with the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2014) on September 19, 2014.

This workshop starts from awareness that modern automatic systems are able to collect huge volumes of data, often with a complex structure (e.g. multi-table data, XML data, web data, time series and sequences, graphs and trees). This fact poses new challenges for current information systems with respect to storing, managing and mining these sets of complex data.

The workshop follows the successful two previous editions (NFMCP 2012 and NFMCP 2013), which were held in conjunction with ECML-PKDD 2012 and ECML-PKDD 2013 respectively, as well as several editions of the international workshop on Mining Complex Data (MCD 2006@IEEE ICDM 2006, MCD 2007@ECML/PKDD 2007, MCD 2008@IEEE ICDM 2008).

Our purpose in this workshop was to bring together researchers and practitioners of data mining who are interested in the advances and latest developments in the area of extracting patterns from complex data sources like blogs, event or log data, medical data, spatio-temporal data, social networks, mobility data, sensor data and streams, and so on.

We received nineteen submissions in several research fields ranging from stream data mining to sequence mining, graph mining, bio-medic, process and music mining. We were able to accept seventeen papers, based on a rigorous reviewing process. Each submission was evaluated by three independent referees. Additionally, the scientific program also featured an invited talk by Thomas Gärtner (University of Bonn and Fraunhofer IAIS) on "Sampling and Presenting Patterns from Structured Data".

We would like to thank all the authors who submitted papers, the invited speaker and all the workshop participants and speakers. We are also grateful to the members of the program committee and external referees for their excellent work in reviewing submitted and revised contributions with expertise and patience. We would like to acknowledge the support of the European Commission through the project MAESTRA - Learning from Massive, Incompletely annotated, and Structured Data (Grant number ICT-2013-612944). A special thank is due to both the ECML PKDD Workshop Chairs and to the members of ECML PKDD organizers who made the event possible.

Annalisa Appice,
Michelangelo Ceci,
Corrado Loglisci,
Giuseppe Manco,

Elio Masciari,
Zbigniew W. Ras.

September 2014.

Organization

Program Chairs

Annalisa Appice	University of Bari "Aldo Moro", Bari, Italy
Michelangelo Ceci	University of Bari "Aldo Moro", Bari, Italy
Corrado Loglisci	University of Bari "Aldo Moro", Bari, Italy
Giuseppe Manco	ICAR-CNR, Rende, Italy
Elio Masciari	ICAR-CNR, Rende, Italy
Zbigniew W. Ras	University of North Carolina, Charlotte, USA & Warsaw University of Technology, Poland

Program Committee

Nicola Barbieri, Yahoo Research Barcelona, Spain
Petr Berka, University of Economics, Prague, Czech Republic
Saso Dzeroski, Jozef Stefan Institute, Slovenia
Stefano Ferilli, University of Bari, Italy
Mohand-Said Hacid, Université Claude Bernard Lyon 1 - UCBL, France
Dino Ienco, IRSTEA, France
Joao Gama, University Porto, Portugal
Dragi Kocev, Jozef Stefan Institute, Slovenia
Mirco Nanni, KDD-Lab ISTI-CNR Pisa, Italy
Apostolos N.Papadopoulos, Aristotle University of Thessaloniki, Greece
Fabrizio Riguzzi, University of Ferrara, Italy
Henryk Rybinski, Warsaw Univ. of Technology, Poland
Jerzy Stefanowski, Poznań Univeristy of Technology, Poland
Herna Viktor, University of Ottawa, Canada
Alicja Wieczorkowska, Polish-Japanese Institute of Information Technology, Poland
Wlodek Zadrozny, University of North Carolina, Charlotte US

Additional Reviewers

Vânia G. Almeida
Paolo Cintia
João Duarte
Massimo Guarascio
Ettore Ritacco

Table of Contents

Sampling and Presenting Patterns from Structured Data

Thomas Gärtner

University of Bonn and Fraunhofer-Institut für Intelligente Analyse und Informationssysteme
Schloss Birlinghoven, 53757 Sankt Augustin, Germany

Abstract. In this talk I will describe some approaches for efficient pattern generation as well as presentation. In particular, I will show pattern sampling algorithms that can easily be extended to structured data and an interactive embedding technique that allows users to intuitively investigate pattern collections.

Location Prediction of Mobile Phone Users using Apriori-based Sequence Mining with Multiple Support Thresholds

Ilkcan Keles, Mert Ozer, I. Hakki Toroslu, Pinar Karagoz, and Salih Ergut[⋆]

Computer Engineering Department
Middle East Technical University, Ankara, Turkey
{ilkcan,mert.ozer,toroslu,karagoz}@ceng.metu.edu.tr

Abstract. Due to the increasing use of mobile phones and their increasing capabilities, huge amount of usage and location data can be collected. Location prediction is an important task for mobile phone operators and smart city administrations to provide better services and recommendations. In this work, we propose a sequence mining based approach for location prediction of mobile phone users. More specifically, we present a modified Apriori-based sequence mining algorithm for the next location prediction, which involves use of multiple support thresholds for different levels of pattern generation. The proposed algorithm involves a new support definition, as well. We have analyzed the behaviour of the algorithm under the change of threshold through experimental evaluation and the experiments indicate improvement in comparison to conventional Apriori-based algorithm.

Keywords: Sequential Pattern Mining, Location Prediction, Mobile Phone Users

1 Introduction

Intensive amounts of basic usage data including base station, call records and GPS records are stored by large-scale mobile phone operators. This data gives companies ability to build their user's daily movement models and helps them to predict the current location of their users. Location prediction systems usually make use of sequential pattern mining methods. One common method usually follows two steps; extract frequent sequence patterns and predict accordingly. These methods mostly use Apriori-based algorithms for the phase of extracting sequence patterns.

Rather than using whole patterns contained in the CDR data implicitly, we need to devise a control mechanism over the elimination of sequence patterns. It is a well known fact that when minimum support gets lower, number of patterns extracted increases, thereby size of prediction sets for the next location of a person gets larger and accuracy of predictions eventually increases. However,

[⋆] AVEA, Istanbul, Turkey

the larger number of patterns causes larger space cost. Conventional technique to prevent space cost explosion is to increase minimum support value. Yet this time, it decreases the number of frequent patterns and the size of the prediction sets dramatically, and this causes to miss some interesting patterns in data. To prevent possible space explosion and not to miss valuable information in data, we propose a modified version of Apriori-based sequence mining algorithm, that works with level-based multiple minimum support values instead of a global one. To the best of our knowledge, this is the first work which uses different minimum support values at different levels of pruning phases of the conventional algorithm.

Normally, the number of levels for Apriori-based sequence mining algorithms is not pre-configured. However, in our case, we consider a predefined number of previous steps to predict the next one. Therefore, we can set the number of levels in Apriori search tree. Moreover, we slightly change the definition of minimum support, which will be defined in the following sections, in our context. We have experimentally compared the performance of the proposed method involving multiple support thresholds in comparison to that of conventional Apriori-based algorithm that uses only a single minimum support value. The experiments indicate that the proposed approach is more effective to decrease the prediction count and memory requirement.

The rest of this paper is organized as follows. Section 2 introduces previous work on location prediction. Section 3 presents the details of the proposed solution. Section 4 gives the information about evaluation metrics and Section 5 presents experimental results of our prediction method. Section 6 concludes our work and points out possible further studies.

2 Previous Work

In recent years, a variety of modification of the minimum support concept in Apriori-based algorithms have been proposed ([2], [3], [4],[5], [6]) for both association rule mining and location prediction problems. In [2], Han and Fu propose a new approach over the conventional Apriori Algorithm that works with association rules at multiple concept levels rather than single concept level. In [3], Liu et al., propose a novel technique to the rare item problem. They define a modified concept of minimum support which is a minimum item support having different thresholds for different items. In [5], Toroslu and Kantarcioglu introduce a new support parameter named as repetition support to discover cyclically repeated patterns. The new parameter helps them to discover more useful patterns by reducing the number of patterns searched. In [6], Ying et al. propose a location prediction system using both conventional support concept and a score value that is related with semantic trajectory pattern in the candidate elimination phase.

In addition to the Apriori-based modifications mentioned above, in [8], Yavas et al. presented an AprioriAll based sequential pattern mining algorithm to find the frequent sequences and to predict the next location of the user. They added a new parameter which is named as maximum number of predictions and it is used to limit the size of the prediction set.

Most of the multiple minimum support concept is based on the rare itemset problem. To the best of our knowledge, this is the first work which uses different minimum support values at the different levels of pruning phases of conventional algorithm. In our previous work on location prediction with sequence mining [7], we broadened the conventional pattern matching nature of sequence mining techniques with some relaxation parameters. In this work, we use some of these parameters introduced in [7].

3 Proposed Technique

3.1 Preliminaries

In this work, we utilized the CDR data of one of the largest mobile phone operators of Turkey. The data corresponds to an area of roughly 25000 km^2 with a population around 5 million. Almost 70% of this population is concentrated in a large urban area of approximately 1/3 of the whole region. The rest of the region contains some mid-sized and small towns and large rural area with very low population. The CDR data contains roughly 1 million users' log records for a period of 1 month. For each user, there are 30 records per day on average. The whole area contains more than 13000 base stations. The records in CDR data contain anonymized phone numbers (of caller and callee or SMS sender and receiver), the base station id of the caller (sender), the time of the operation.

Unnecessary attributes in CDR data, such as city code, phone number etc., are filtered out and date and time information are merged into a single attribute which is used to sort data in temporal order. After sorting, we created sequences of fixed-length corresponding to user's daily movement behavior.

A *sequence* is an ordered list of locations which is expressed as $s < i, .., j >$, where i is the starting location and j is the last location in the sequence. A sequence of length k is called *k-sequence.*

In Apriori-based sequence mining, the search space can be represented as a hash tree. A *path* in the tree is a sequence of nodes such that each node is the prefix of the path until the root and, for each node, its predecessor is the node's parent. p<a..b> expresses a path starting with node a and ending with node b.

We say that a path p is equal to a sequence s, denoted by $p = s$, if the length of path p and sequence of s are equal and there is one to one correspondence between the locations of s and the nodes of p.

We say that a sequence $s < s_1, s_2, ..., s_n >$ is *contained in* another sequence $s' < s'_1, s'_2, ..., s'_m >$ if there exists integers $i_1 < i_2 < ... < i_n$ such that $s_1 = s'_{i_1}, s_2 = s'_{i_2} ... s_n = s'_{i_n}$.

A sequence s is a subsequence of s' if s is contained in s' and it is denoted by $s \subseteq s'$.

3.2 Apriori-based Sequence Mining Algorithm with Multiple Support Thresholds (ASMAMS)

To build a model which aims to predict the next location of the user, we developed a recursive hash tree based algorithm namely Apriori-based Sequence Mining Algorithm with Multiple Support Thresholds (ASMAMS). This algorithm constructs level based models i.e. hash trees whose nodes contain corresponding base station id and frequency count of the sequence corresponding to the path up to this node.

The main novelty of the algorithm in comparison to the conventional algorithm is the level based support mechanism with a new level-based support definition. In contrast to previous approaches that aim to extract all frequent sequences, in this work, we focus on predicting the next item in a sequence. Therefore, we defined a level-based support in order to keep track of the relations between levels. Conventionally, support of a given sequence pattern is defined as the ratio of the number of the sequences containing the pattern to the number of all sequences in the dataset. In ASMAMS, support of an n-sequence is defined as the ratio of the count of a given sequence s to the count of the parent sequence with length *(n-1)*.

$$support(s) = \frac{\#\ of\ occurrences\ of\ the\ sequence\ s\ with\ length\ n}{\#\ of\ occurrences\ of\ prefix\ of\ sequence\ s\ with\ length\ (n-1)}. \quad (1)$$

The following parameters will be used by ASMAMS:

- *levelCount:* The height of the hash tree.
- *currentLevel:* Current level throughout the construction of the hash tree.
- *supportList:* List of minimum support parameters for each level.
- *sequences:* A set of fixed-length location id sequences.
- *tree:* Hash tree where each node stores the location id and the count of sequence represented by a path from root to this node.
- *tolerance:* Length tolerance of rule extraction phase.

ASMAMS algorithm has three phases which are model construction, rule extraction and prediction. As given in Algorithm 1, model construction phase is divided into two sub-phases: tree construction and pruning.

In the tree construction phase, the data is read sequentially, and new level nodes are added to the corresponding tree nodes. For instance, assume that we are constructing the fourth level of the tree and we have <1,2,3,4> as the sequence. If <1,2,3> corresponds to a path in the input tree, 4 is added as a leaf node as the prefix of this path with count 1. If we encounter the same sequence, the algorithm only increments the count of this node. If the current tree does not contain <1,2,3>, then it is not added to the tree. The construction algorithm is given in Algorithm 2.

In the pruning phase, constructed model and the corresponding minimum support value are taken as parameters. In this phase, initially we calculate leaf nodes' support values. If it is below the minimum support value, it is removed from tree, otherwise no action is taken.

Algorithm 1 ASMAMS Model Construction Phase

```
Input: $sequences$,$levelCount$,$supportList$,$currentLevel \leftarrow 1$
Output: $tree$
1: function buildModel($sequences$, $levelCount$, $currentLevel$, $supportList$, $tree$)
2:     $constructTree(sequences, tree, currentLevel)$
3:     $pruneTree(tree, currentLevel, supportList[currentLevel])$
4:     if $currentLevel \neq levelCount$ then
5:         $buildModel(levelCount, currentLevel + 1, supportList, tree)$
6:     end if
7: end function
```

Algorithm 2 ASMAMS Tree Construction Phase

```
Input: $sequences$, $tree$, $currentLevel$
Output: $tree$
1: function constructTree($sequences$, $tree$, $currentLevel$)
2:     for all $s\langle l_1..l_{currentLevel}\rangle \in sequences$ do
3:         if $\exists p\langle root..leaf\rangle \in tree$ s.t $p = s$ then
4:             $leaf.count = leaf.count + 1$
5:         else
6:             if $\exists p\langle root..leaf\rangle \in tree$ s.t $p = s\langle l_1..l_{currentLevel-1}\rangle$ then
7:                 $insert(tree, leaf, l_{currentLevel})$ //add $l_{currentLevel}$ as a child of $leaf$
8:                 $l_{currentLevel}.count = 1$
9:             end if
10:        end if
11:    end for
12: end function
```

Rule Extraction In the rule extraction phase, the algorithm extracts rules from the hash tree built in model construction phase with respect to a tolerance parameter. If tolerance parameter is set to 0, the algorithm extract rules, whose left-hand side contains ($levelCount - 1$)-sequence and right-hand side contains the output level location, from the $levelCount$-sequence s as follows:

$[s_1, s_2, ..., s_{levelCount-1} \rightarrow s_{levelCount}]$.

If tolerance is greater than 0, the algorithm extract rules until the left-hand sides of the rules have the length of $levelCount - (tolerance + 1)$ as shown in Algorithm 3.

Prediction In the prediction phase, we use set of rules constructed by rule extraction phase to predict user's next location. The prediction algorithm takes a sequence as input and returns a list of predicted locations.

The algorithm firstly checks whether rules with length of $levelCount$ is contained in the given sequence. In that case, the right-hand side of the rules constitute the prediction set. If the rules of length $levelCount$ are not contained in the given sequence, then it checks whether the rules of length $levelCount - 1$ are contained in the given sequence. This continues until the rules are contained in the sequence or until the tolerance parameter is reached but no output is

Algorithm 3 ASMAMS Rule Extraction Phase

Input: *tree*, *levelCount tolerance*
Output: *ruleSet*

```
1: function ruleExtraction($tree$, $levelCount$, $tolerance$)
2:     for all $s{<}s_1, s_2, ..., s_{levelCount}{>} \in tree$ s.t. $length(s) = depth(tree)$ do
3:         for $t = 0$ to $tolerance$ do
4:             $subSequencesSet \leftarrow t$-deleted subsequences of $s{<}s_1, ..., s_{levelCount-1}{>}$
5:             for all subsequence $s' \in subSequencesSet$ do
6:                 $ruleSet \leftarrow ruleSet \cup \{s' \rightarrow s_{levelCount}\}$ //Add new rule to $ruleSet$
7:             end for
8:         end for
9:     end for
10: end function
```

produced. The detailed algorithm of prediction phase can be found in Algorithm 4.

Algorithm 4 Prediction Algorithm

Input: *sequence*, *ruleSet*, *levelCount*, *tolerance*
Output: *predictionSet*

```
1: function predict($sequence$, $ruleSet$, $levelCount$, $tolerance$)
2:     for $t = 0$ to $tolerance$ do
3:         for all $rule \in rules$ of length $levelCount - t$ do
4:             if $lhs(rule) \subseteq sequence$ then
5:                 $predictionSet \leftarrow predictionSet \cup \{rhs(rule)\}$
6:             end if
7:         end for
8:         if $predictionSet \neq \emptyset$ then
9:             break
10:        end if
11:    end for
12:    return $predictionSet$
13: end function
```

Running Example In this example, we set level count to 5 and minimum support list to [0.16, 0.5, 0.5, 0.66, 0] and we use the sample sequences shown in the Table 1.

In the first level, the data is traversed sequentially and the first location ids in the sequences are added to the hash tree together with their counts. Then in the pruning phase, their support values are calculated and nodes 2 and 3 are pruned since their support fall below the given minimum support 0.16. In the second level, 2-sequences are added to the hash tree with their counts. After support values are found, the nodes <5,6>, <5,8> and <5,11> are pruned since their support values are 0.33 and falls below the given minimum support 0.5. The resulting hash trees can be seen in Figure 1.

Table 1: Example Sequences

id	sequence	-	id	sequence
1	<1, 2, 3, 4, 5>		7	<4, 7, 11, 12, 15>
2	<1, 2, 3, 4, 6>		8	<4, 7, 11, 10, 9>
3	<1, 2, 3, 4, 5>		9	<5, 6, 11, 10, 9>
4	<2, 3, 4, 7, 8>		10	<5, 8, 9, 10, 11>
5	<3, 4, 7, 9, 10>		11	<5, 11, 10, 9, 4>
6	<4, 7, 11, 12, 13>		12	<1, 2, 3, 4, 5>

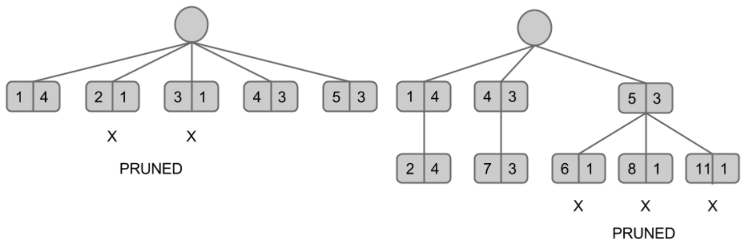


Fig. 1: Hash tree at the end of the first level (left), Hash tree at the end of the second level (right)

In the third level, 3-sequences are added to the hash tree. None of the nodes are pruned in this level, since the support values are all 1. In the fourth level, after 4-sequences are added to the hash tree, the node <4,7,11,10> is pruned as it does not have the required support. In the final level (which is the last level of the hash tree), 5-sequences are added to the hash tree. Since the minimum support value for this level is 0, there is no pruning. The resulting hash tree can be seen in Figure 2.

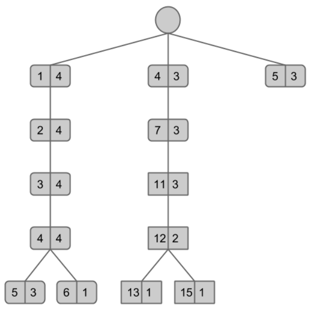


Fig. 2: Hash tree at the end of the final level

Using the hash tree constructed by model construction phase which is shown in Figure 2, the rules are extracted according to the *tolerance* parameter. If the tolerance parameter is 0, the following rules are extracted: $[1, 2, 3, 4 \rightarrow 5]$, $[1, 2, 3, 4 \rightarrow 6]$, $[4, 7, 11, 12 \rightarrow 13]$, $[4, 7, 11, 12 \rightarrow 15]$.

In this case, for a sequence of $<1, 2, 3, 4>$, the algorithm gives the output of 5 and 6. However, for a sequence of $<1, 2, 8, 3>$, the algorithm does not generate any output. If the tolerance parameter is 1, the following extra rules are extracted:$[1, 2, 3 \rightarrow 5]$, $[1, 2, 4 \rightarrow 5]$, $[1, 3, 4 \rightarrow 5]$, $[2, 3, 4 \rightarrow 5]$, $[1, 2, 3 \rightarrow 6]$, $[1, 2, 4 \rightarrow 6]$, $[1, 3, 4 \rightarrow 6]$, $[2, 3, 4 \rightarrow 6]$, $[4, 7, 11 \rightarrow 13]$, $[4, 7, 12 \rightarrow 13]$, $[4, 11, 12 \rightarrow 13]$, $[7, 11, 12 \rightarrow 13]$, $[4, 7, 11 \rightarrow 15]$, $[4, 7, 12 \rightarrow 15]$, $[4, 11, 12 \rightarrow 15]$, $[7, 11, 12 \rightarrow 15]$.

By using the tolerance parameter, for a sequence of $<1, 2, 8, 3>$, the algorithm generates the output of 5 and 6, since the left side of the rule $[1, 2, 3 \rightarrow 5]$ and $[1, 2, 3 \rightarrow 6]$ are contained in the given sequence.

4 Evaluation

For the experimental evaluation, CDR data obtained from one of the largest mobile phone operators in Turkey has been used. A quick analysis shows that around 76% of the users next location is their current location. We take this value as the baseline for our experiments. For evaluation, we extract sequences from raw CDR data set and try to predict the last element of the sequence using the previous ones. After trying several lengths, we have determined that 5-sequences (i.e., using a given 4-sequence, try to predict the next element of the sequence) produces the highest accuracy values. Therefore, we have used 5-sequences extracted from data set, both for training and testing, by using k-fold cross validation in order to assess the quality of predictions made. As training phase, we run ASMAMS on fixed length sequences to build the sequence tree. At the testing phase, for each test set sequence Algorithm 4 introduced in the section 3.5 has been applied and the result of the prediction is compared against the actual last element of the test set sequence. These results are used in the calculations of the evaluation metrics which are introduced below.

Accuracy metric is used for evaluating the number of correctly predicted test set sequences. It simply can be defined as the ratio of true predicted test sequences to the total number of test sequences. However, for some test cases, there may be no relevant path in the tree for test sequence which means either no such training sequence is come up or it is removed from the tree in one of the pruning phases. The first accuracy metric, g-accuracy (general accuracy), is the ratio of number of correctly predicted test sequences to the number of all test sequences. The second one, p-accuracy (predictions' accuracy), is the ratio of the number of correctly predicted test sequences to the number of all test sequences able to be predicted. In the first form of accuracy calculation, the accuracy result superficially drops for cases that no prediction is able to be performed. These accuracy measures have been described in more detail in our earlier work [7].

Memory Requirement metric measures the relative peak RAM requirement during the algorithm's execution. All memory requirement values are projected to the range [0-100], where 100 represents the maximum memory utilization.

Prediction Count metric is used to evaluate average size of the prediction set in correctly predicted test sequences.

Score is introduced since there are 4 different parameters that we want to optimize. It is used for evaluating general performance of our model by combining above metrics into a single one. This metric is only used to determine the parameters for the optimal model. It is defined as a weighted sum of *g-accuracy*, *p-accuracy*, *memory requirement*(mem_req) and *prediction count*(pred_count) in Equation 2.

$$Score = w_1 * \textit{g-accuracy} + w_2 * \textit{p-accuracy} + w_3 * \textit{(100-mem\_req)} + w_4 * \textit{(100-pred\_count)}. \quad (2)$$

Considering the importance of the parameters the weights are set as follows; w1 = 0.6, w2 = 0.1, w3 = 0.1 and w4 = 0.2.

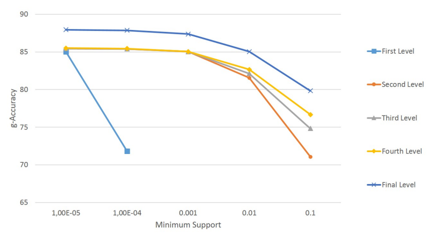
5 Experimental Results

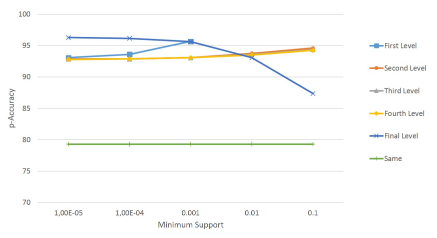
For the experiments, we have used 5-sequences (i.e. level count in Algorithm 2 is set to 5), after trying longer and shorter sequences. While shorter sequences, such as 4-sequences or 3-sequences, were superficially increasing prediction count, longer sequences, such as 6-sequences, were decreasing g-accuracy sharply, even though p-accuracy was increasing, since the number of predictable sequences was quickly decreasing. Therefore, 5-sequences seemed as the best for the data in hand, and shorter or longer sequences' results were not useful.

After determining the sequence length and level count for experiments, we first narrow down our search space by setting our support values to a set $\{10^{-5}$, 10^{-4}, 10^{-3}, 10^{-2}, $10^{-1}\}$ for each level. We have used the score parameter introduced above to determine this best support list as [10^{-5},10^{-3}, 10^{-3}, 10^{-3}, 10^{-2}]. Then we have tried all possible non-decreasing combinations as list of support parameters. For every level, we fixed other levels' support values to the support values of the best model and we present results of changing this level's minimum support value according to evaluation metrics. Same percentage value refers to the ratio of being in the same location as previous location and is included in the figure to show the improvement provided by ASMAMS.

In a set of experiments, we have analyzed the effect of the minimum support parameter for all levels. In order to do that, for each level, the experiments are performed with the support values explained above and other levels' support parameters are set to the optimal values. In addition, the tolerance parameter is fixed to 0 for first set of the experiments.

As it can be seen from the Figure 3, for all levels, g-accuracy drops as the minimum support increases. However, this drop is much sharper in the first level.

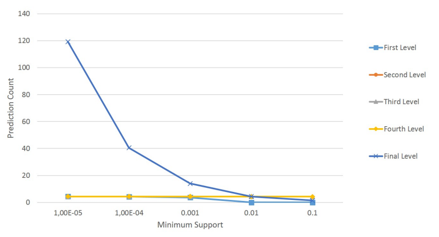


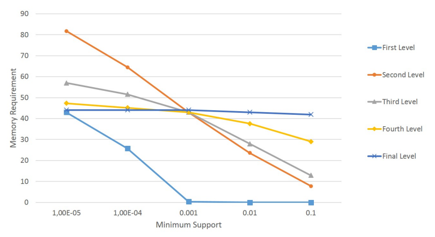


(a) (b)

Fig. 3: Change of g-Accuracy & p-Accuracy

Although, p-accuracy also shows the same trend in the first level, it shows slight increase in intermediate levels, and then, there is also a small drop in the final level. Figure 3 also shows the percentages of locations which are exactly the same as the previous ones for all the experiments as well. Our p-accuracy results show that, the correct prediction (of p-accuracy) can be increased even above 95% with our model.





(a) (b)

Fig. 4: Change of Prediction Count & Memory Requirement

Similar trends can be observed for the prediction count parameter. Sharp drops occur in the first level as the minimum support value increases. However, for intermediate levels these drops are almost negligible. Again, in the final level, prediction count decreases much faster also. Figure 4 shows that the prediction count values are at acceptable levels.

The amount of the drop in the memory requirement as the minimum support value increases slows down with the increase of the levels. In the final level, there is almost no drop in the memory requirement. Especially in the first level, since most sequences are pruned with high minimum support requirement, the memory requirement drops very quickly.

In addition to above mentioned experiments, we have also applied standard AprioriAll algorithm [1]. The main drawback of AprioriAll algorithm is the size of the prediction set. In order to obtain high accuracy results (g-accuracy) as in our model, the minimum support value must be chosen as a very small value (even zero), so that we can keep as much sequences as possible. However, this results in high prediction count as well as increasing the memory requirement. The accuracy obtained when no minimum support value is given is the upper bound that can be achieved with sequence matching approach. However, for that setting the memory requirement is also the maximum, since the hash-tree keeps all sequences without any pruning. As expected, this maximum accuracy can be obtained only with a very high prediction count, which is more than 133. Since this is unacceptably high, we tested AprioriAll with a non-zero, but very small minimum support value. This resulted slight decrease in accuracy, while dropping the prediction count and the memory requirement significantly with pruning of large portion of hash-tree. Even though the memory requirement has dropped a lot to a very good level, the decreased value of prediction count still stayed unacceptably high value, which is almost 40. Further increases in minimum support values had dropped the accuracy levels to around and below baseline levels. Therefore, they are not acceptable either. However, with ASMAMS we have achieved almost the same accuracy levels of the best and optimal AprioriAll accuracy values with a very low prediction count value, which is 4.43, with a memory requirement less than the half of the optimal (and maximal) results of AprioriAll setting. In addition to this, we have applied ASMAMS with a tolerance value 1 and we achieved a general accuracy of 88.68 with nearly same prediction count. We have also applied ASMAMS with a tolerance value 2, however, since no prediction ratio is really low, it did not produce any improvement for our dataset. These results are summarized in Table 2.

Table 2: The results for ASMAMS and AprioriAll methods

G-Accuracy	P-Accuracy	Mem. Req.	Pred. Count	No Output Ratio	Description
88.68	89.44	44	4.42	0.8%	ASMAMS Min. Sup. List: [1e-5.1e-3.1e-3.1e-3.1e-2] Tolerance:1
85.04	93.08	44	4.43	8.6%	ASMAMS Min. Sup. List: [1e-5.1e-3.1e-3.1e-3.1e-2] Tolerance:0
51.47	88.66	0.01	1.29	51.47%	ApprioriAll Min. Sup: 1e-5
86.32	94.15	9.76	39.42	8.32%	ApprioriAll Min. Sup: 1e-8
89.82	95.38	100	133.48	5.84%	ApprioriAll Min. Sup: 0

6 Conclusion

In this work, we present an Apriori-based sequence mining algorithm for next location prediction of mobile phone users. The basic novelty of the proposed algorithm is a new, level-based support definition and the use of multiple support thresholds, each for different levels of pattern generation that corresponds to generation of sequence patterns of different lengths. The evaluation of the method is conducted on CDR data of one of the largest mobile phone operators in Turkey. The experiments compare the performance of the proposed method in terms of accuracy, prediction count and space requirement under varying thresholds for each level. Actually, these experiments serve for determination of the best minimum support list for each level to obtain the highest accuracies, as well. We have also compared the performance with conventional method involving a single support threshold. We have observed that our method ASMAMS produces almost the optimal accuracy results with very small prediction sets, whereas the same accuracy can be obtained by AprioriAll with very low support thresholds and much larger prediction sets. Considering that there are more than 13000 different locations, the prediction sets' sizes, such as 4, obtained by ASMAMS with almost optimal accuracy can be considered as quite useful result for the mobile phone operator.

As the future work, we aim to extend this study by adding a region based hierarchy to this model in order to increase prediction accuracy.

Acknowledgements. This research was supported by Ministry of Science, Industry and Technology of Turkey with project number 01256.STZ.2012-1 and title "Predicting Mobile Phone Users' Movement Profiles".

References

1. Agrawal R, Srikant R. Mining sequential patterns. In: Proc of Int Conf Data Engineering. Taipei: IEEE Computer Society, pp. 3–14. (1995)
2. Han, J., Fu, Y.: Discovery of Multiple-Level Association Rules from Large Databases. In: Proc. VLDB '95, pp. 420–431. (1995)
3. Liu, B., Hsu, W., Ma, Y.: Mining association rules with multiple minimum supports. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '99). ACM, New York, NY, USA, pp. 337–341. (1999)
4. Li, H., Chen, S., Li, J., Wang, S., Fu, Y.: An improved multi-support Apriori algorithm under the fuzzy item association condition. In: International Conference on Electronics, Communications and Control (ICECC '11), pp. 3539–3542, 9-11 Sept. (2011)
5. Toroslu, H., Kantarcioglu M.: Mining Cyclically Repeated Patterns. In: Springer Lecture Notes in Computer Science 2114, p. 83. (2001)
6. Ying, J., J., Lee, W., Weng, T., Tseng, V. S.: Semantic trajectory mining for location prediction. In: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '11), Divyakant Agrawal, Isabel Cruz, Christian S. Jensen, Eyal Ofek, and Egemen Tanin (Eds.). ACM, New York, NY, USA, 34-43. (2011)
7. Ozer, M., Keles, I., Toroslu, H., Karagoz, P.: Predicting the change of location of mobile phone users. In: Proceedings of the Second ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems (MobiGIS '13), Chi-Yin Chow and Shashi Shekhar (Eds.). ACM, New York, NY, USA, 43-50. (2013)
8. Yavas, G., Katsaros, D., Ulusoy, O.: A Data Mining Approach for Location Prediction in Mobile Environments. In Data Knowl. Eng. 54, pp. 121-146. (2005)

Temporal Constraints Improve Inexact Subgraph Matching in Undirected Networks

Ursula Redmond* and Pádraig Cunningham

School of Computer Science and Informatics,
University College Dublin, Ireland

Abstract. Subgraph matching is the process of identifying embeddings of a query graph in a network graph. Applications range from locating suspicious patterns in financial transaction networks to finding patterns of disease spread in epidemic networks. Sometimes the match must be inexact, since the data may be incomplete due to noise, or a user may seek a range of graph structures similar to a query. However, inexact subgraph matching increases the number of potential matches and computational expense. If we require the matches to be temporally constrained, processing time can be reduced. An embedding is temporally constrained if adjacent interactions occur close in time, simple paths are time-respecting, and cycles are mostly time-respecting. To validate our approach, we apply our algorithm to undirected face-to-face contact networks collected from the SocioPatterns project. Our experimental results demonstrate that our algorithm identifies query embeddings notably faster when temporal information is incorporated.

1 Introduction

Many large network data sets are composed of sets of interactions that represent some process or functionality within the network [10]. If a specific query graph – made up of a combination of interactions – is of interest, then subgraph matching may be employed to identify embeddings of the query graph in the network. The structure of the query graph sought depends on the type of network being searched. In a financial transaction network, a query graph might represent a pattern observed in fraud. In an epidemic network, a query with a cascade structure might represent disease spreading. Dense structures in a communication network might represent communities. In a face-to-face contact network, a query graph might represent a group of people facing each other, and later perhaps mingling with other people standing close by.

However, network data which is gathered experimentally may be incomplete. In a face-to-face contact network, devices which record contact may miss some interactions, or record interactions erroneously. Also, a user seeking a particular structure may want to examine a range of similar structures. Inexact subgraph

* This work was supported by Science Foundation Ireland [08/SRC/I1407, SFI/12/RC/2289].

matching is a more appropriate approach for this problem. This method seeks embeddings of the query graph in the network which allow for some differences in the structure. A side-effect of this more general approach to subgraph matching is that the number of potential matches increases, which may be a problem in large networks.

Increasingly, temporal information is available with network data sets, which encodes the time at which interactions took place. In this work, we use temporal information to constrain the inexact subgraph matching process. The traditional approach of time-slicing first uses a sliding window to isolate data that occur between a start and end time, then examines the resultant data as if it were static. Time-slicing risks arbitrarily cutting potential contagions in a network, whose originating and terminating individuals may not be present in a time-slice, as an artifact of the time window choice. To avoid this problem, we enforce temporal constraints on a subgraph at the level of its interactions. In this way, if a contagion exists in the network, the entire structure can be found if the pair-wise interactions making it up obey the temporal constraints.

We require that embeddings of a query graph in a network are temporally constrained. For an embedding to be temporally constrained, three criteria must be met. Firstly, adjacent interactions in the network must occur close in time, where this closeness is specified by a time-delay threshold. Secondly, all cycles in the embedding must be composed of interactions that follow each other in time (one reversal of the sequence is inevitable, and allowed). Thirdly, interactions on all simple paths (which are not part of cycles) must follow each other in time. These restrictions reduce the number of embeddings, speeds up the search process, and ensures that returned embeddings are meaningful in a temporal context.

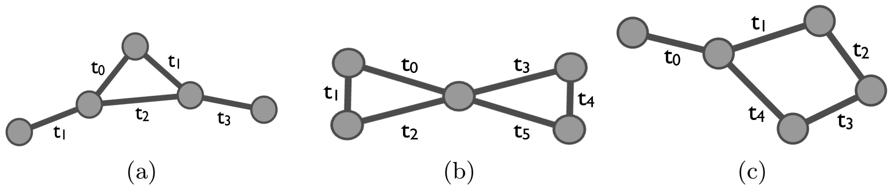


Fig. 1: Three examples of query graph embeddings in an undirected network. All adjacent interactions occur close enough in time, if we set the time-delay threshold to 6 time units. The interactions in all paths follow each other in time, except for cycles, for example in Fig. 1a, $t_0 \leq t_1 \leq t_2$, but $t_2 > t_0$.

A set of query graph embeddings are shown in Fig. 1. Time-stamps are shown on the interactions, to demonstrate the temporal constraints. If noise were present in the network, then some of the interactions in the embeddings might be missing, or extra interactions might be present. In that case, an inexact matching algorithm would be required, to locate the interactions among individuals who had been in contact in a way similar to that sought.

We apply our inexact matching algorithm to two undirected face-to-face contact networks which were collected during the SocioPatterns project [6]. Our approach is demonstrated to be faster when using temporal information than in the static case, for a range of query graphs and measures of inexactness. Since the interactions present within the temporally constrained embeddings occur close in time, the subgraphs found are a good approximation for contact as it occurred over specific time periods. So, although the number of embeddings returned is smaller, the meaning of those embeddings is easier to interpret. As the matches are allowed to vary more from the query structure, our temporal approach performs orders of magnitude faster than if the networks searched were static.

2 Related Work

This work draws upon the areas of temporal network analysis and inexact graph and subgraph matching. We extend inexact subgraph matching to account for temporal information in networks.

2.1 Inexact Graph and Subgraph Matching

Inexact graph matching is a generalization of conventional graph isomorphism. The problem is also know as approximate graph matching [18], or error-tolerant graph matching. The process of inexact graph matching is usually thought of as a transformation from one graph into another, via a sequence of edits. Each edit operation has a cost, which depends on the differences seen so far, between the two matched graphs. These edits include the insertion, deletion and relabeling of the nodes and edges, which are required to transform one graph into the other.

Given a visual representation of a system, such as in scene analysis, switching theory or chemical structure analysis, a graph representing relations is often useful. The graph patterns may be deformed, so inexact graph matching comes into play. Tsai and Fu [16] guide the search for matches using an ordered-search algorithm. This finds embeddings that most closely match the query graph first. Another application of inexact graph matching is to match chemical compounds, in which the labeling needed to identify certain molecules is unknown. An algorithm was proposed by Hofer *et. al.* which accounts for wildcard labeling in the match [4]. Thus, when the label on a molecule is not known, it can assume a label which makes sense based on chemical expert knowledge.

Inexact subgraph matching generalizes inexact graph matching. Instead of checking that two entire graphs match inexactly, an embedded subgraph (or set of subgraphs) in the larger graph is sought that inexactly matches some query graph. Such an error-correcting subgraph isomorphism algorithm was proposed by Tsai and Fu [17]. The G-Ray algorithm of Tong *et. al.* [15] aims to find subgraphs that match a query pattern closely, by allowing indirect paths between attributed graph entities. The results are returned in order of their "goodness", as defined by the authors. Tian and Patel developed TALE [14], a tool which uses an index structure to speed up the approximate matching of subgraph queries in a database of graphs. This technique identifies nodes that are important in the match, based on their position in the graph structure, then extends those matches.

2.2 Temporal Network Analysis

Many techniques for static graph analysis must be revised for graphs which contain temporal information. A comprehensive review is provided by Holme *et al.* [5]. Kempe *et al.* [7] define a *time-respecting* path as a sequence of contacts that occur at non-decreasing times.

Reachability graphs show which nodes are reachable from a single root node [11]. In a reachability graph, there must be a time-respecting path between nodes i and j for a directed edge to exist between them. Bearman *et al.* [2] analyze the reachability graph within a dating network of high-school students. A *time-respecting subgraph* [13] is a generalization of a reachability graph, similar to how a subgraph generalizes a tree structure.

The lifespan of a piece of information may be specified by a time window [19]. This measures the time between two consecutive communications. The claim is that the closer in time the contacts take place, the more likely they are to be about the same topic. In the same way, the *relay time* of an interaction describes the time taken for a newly infected node to spread the infection further via subsequent interactions [8]. When matching temporally constrained subgraphs, we require that consecutive interactions occur within a specified time. A cascade models the spread of information through a network, for example the adoption of new ideas or products [9]. The importance of *time-constrained* cascades is emphasized for understanding contagion [1].

3 Methods

We introduce some graph theoretic notation. A graph is an abstraction of a network. A graph $G = (V, E)$ is composed of a set of vertices (nodes) V and a set E of pairs of nodes called edges. The graph $H = (W, F)$ is a subgraph of G if W is a subset of V and F is a subset of E, such that the nodes of each edge in F are in W. The number of edges incident to a node v is called the degree of v. The *neighbors* of v are nodes which are connected to v via an edge. A pair of edges is adjacent if they share a node. Given that time is encoded as an explicit part of our network representation, instead of referring to an "edge" between two nodes, we use the term "interaction" to specify a triplet, composed of two nodes and their time of contact. We define an undirected temporal graph as follows:

Definition 1. *An undirected temporal graph G consists of a set V of nodes and a set E of pairs of nodes representing interactions. An interaction $e_i \in E$*

is represented by a tuple $e_i = (\{u_i, v_i\}, t_i)$, in which u_i and v_i comprise an unordered set of nodes, and t_i is the initiation time of the interaction.

3.1 Inexact Subgraph Matching

A key feature of our algorithm is that node labels are not required. In large contact networks, labels are not necessarily useful as an aid for finding interesting interaction patterns, since the people in the network are not known *apriori*. This differs from networks of chemical compounds or protein interactions, in which the identification of the entities may be very important. The challenge with excluding node labels for inexact subgraph matching is that there are no obvious starting nodes for the matching process, and the labels cannot be used to guide the search.

We extend an implementation of the VF2 algorithm [3] from the NetworkX Python library [12]. $G1$ represents the large graph in which a query graph $G2$ is sought. The matching process is described by a state space representation. Given an intermediate state s, the mapping is extended by first computing candidate node pairs (one node each from $G1$ and $G2$), then testing their syntactic and semantic feasibility as matching nodes.

Syntactic feasibility is based on topology. We define the *cost* of a match to be the number of edits required to transform the query graph $G2$ into the embedded subgraph of $G1$ in question. This cost is computed dynamically. In accumulating the cost of a match, the edits we allow relate to interactions. From a state s, we calculate the cost of adding the next candidate node pair ($G1\_node$, $G2\_node$), and add that to the current cost. If the accumulated cost exceeds a threshold, θ, the match is discarded.

The difference in the number of self-loops incident to $G1\_node$ and $G2\_node$ adds to the cost of the match. There may also be effects on neighbors of the new nodes. For $G1$, we check each neighbor of $G1\_node$ which is also in the partial mapping of s. If the neighbors of $G1\_node$ found in $G1$ do not have counterparts in $G2$ in the partial mapping, we increment the cost by one. If the number of interactions between each of these neighbors and $G1\_node$ in $G1$ differs from the number of interactions between their counterparts and $G2\_node$ in $G2$, we increment the cost by the difference. The same check is performed from the perspective of $G2$.

We do not allow edits related to nodes. If we allow extra or fewer nodes from $G1$ to participate in a given match, there will be many more potential matches, since the search is not limited through the use of node labels. If required, the user can add or remove nodes from the query before the search is performed. This is an easier task for the user than specifying every combination of interactions that are of interest, which is the problem solved by our algorithm.

3.2 Temporal Constraints

Instead of node labels, we constrain the search space using temporal information. We insist that the embedded subgraphs returned are connected and temporally constrained. There are three types of temporal constraint that we require. These relate to adjacent interactions, paths and cycles.

Definition 2. *Let e_i and e_j be interactions in an undirected temporal graph. The interactions are temporally constrained if they are adjacent and $|t_j - t_i| \leq d$, for some threshold d.*

Definition 3. *A time-respecting path between two nodes v and w in an undirected temporal graph is a finite alternating sequence $v = v_0, e_1, v1, e2, ..., e_n, v_n = w$ of non-repeating nodes and interactions such that the ordered sequence of interaction times is either monotonically increasing or decreasing.*

Definition 4. *A temporally constrained cycle is a path between a node and itself, with no repeating nodes or interactions, such that the path is time-respecting but for one inversion.*

To see why we allow exactly one inversion, consider the following scenario, in which we view a face-to-face contact network as a proxy for communication. At time t_0, individuals i and j communicate with each other. At time t_1, individual j communicates with individual k. There is potential for a piece of information to have been transferred from i to k. Now suppose that k communicates with i at time t_2, dispensing that same piece of information. We can say that a flow of information occurred from i to j to k and back to i, even though the adjacent interaction sequence at times t_2, t_0 is not time-respecting.

Definition 5. *A connected temporally constrained subgraph $S = (V', E')$ of a temporal graph $G = (V, E)$ is a subgraph such that*

- *every adjacent interaction pair is temporally constrained*
- *all cycles are temporally constrained*
- *all simple paths are time-respecting, unless they are connected to a cycle; then the path subset formed from the non-cycle portion of the path and only one adjacent interaction from the cycle must be time-respecting.*

The first condition in Definition 5 is illustrated in Fig. 2. The adjacent interactions share the central node. If, for example, the time delay threshold d was set to four time units, then the interactions all occur within time d of each other. The second condition is shown in Fig. 3. Every cycle must be temporally constrained. The third condition, as shown in Fig. 4, requires that a simple path is time-respecting. In the situation where a simple path intersects with a cycle, as in Fig. 5, one interaction from the cycle must be time-respecting with respect to the non-cycle portion of the path. This ensures that some method of information flow is possible within the subgraph as a whole. In the illustrated example, information communicated via the path may influence information spread within the cycle, since the temporal sequence t_1, t_2 between the path and cycle is time-respecting.

Within the VF2 algorithm, there is an option to test for semantic feasibility during the matching process. In our setting, we use the dates on which

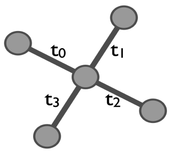


Fig. 2: In a temporally constrained subgraph, all adjacent interactions must occur within time d of each other. Thus, if $t_0 \leq t_1 \leq t_2 \leq t_3$, then $t_3 - t_0 \leq d$ must be true.

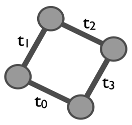


Fig. 3: A temporally constrained cycle, composed of a time-respecting path, which starts and ends on the same node. The interactions incident to this node constitute the only permissible inversion of the time-respecting sequence. In this cycle, $t_0 \leq t_1 \leq t_2 \leq t_3$, but $t_3 > t_0$.

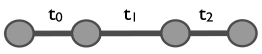


Fig. 4: A time-respecting path, in which no nodes or interactions are repeated, and the sequence of interactions occurs in a monotonically increasing order. Here, $t_0 \leq t_1 \leq t_2$.

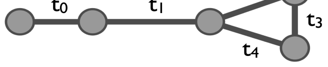


Fig. 5: A time-respecting path which is connected to a cycle. The non-cycle portion of the path includes the interactions at t_0 and t_1. The cycle includes interactions at t_2, t_3 and t_4. There are two interactions from the cycle which are adjacent to the non-cycle path. If we take the interaction at t_2, then the sequence t_0, t_1, t_2 is time-respecting.

the interactions occur to test that the subgraph embedded in $G1$ is temporally constrained. Since we aim to find subgraphs in a network that are temporally constrained and have the topology of $G2$, we assume that $G2$ is temporally constrained by construction and thus do not need to check this at runtime.

Since our algorithm uses a recursive backtracking approach, we explain the semantic feasibility tests with regard to the new candidate node - $G1\_node$ - in the embedding, at some given level of recursion. The first test checks that all interactions adjacent to $G1\_node$ occur within time d of each other, as in Fig. 2. To do this, we use a list *neighbors* containing the neighboring nodes of $G1\_node$ in $G1$ which are part of the current mapping. The list *dates* contains the dates, in increasing order, on which contacts between $G1\_node$ and the nodes in *neighbors* were made. If the difference between the latest and earliest date does not exceed d, we proceed to the next test.

To check that all cycles in the embedding are temporally constrained, we first extract all cycles from the embedding. For each cycle, we construct a new list *dates*, which stores the date between each interaction in the cycle. We step through each date, testing if the sequence is monotonically increasing. We count the inversions in the sequence. We then step through the dates again, this time testing if the sequence is monotonically decreasing, and counting inversions. If the number of inversions in one of the two orderings does not exceed one, we proceed to the next test.

The final test involves simple paths. We start by computing all simple paths between $G1\_node$ and every other node in the embedding. For each path, we check the length of the portion of the path which is not part of any cycle in the embedding. If no cycle intersects the path, we test that the path is time-respecting. If it is, then $G1\_node$ is feasible for the match. Otherwise, if the length of the non-cycle path is two or more, but continues into a cycle, then we need to do more testing. Thus, we identify the two interactions in the cycle which are adjacent to the path. If, with one of these interactions, the non-cycle path remains time-respecting, then $G1\_node$ is feasible for the match.

When a candidate node is included in a potential embedding of $G2$ in $G1$, the three conditions in Definition 5 must be fulfilled. If any of the tests fail, the candidate node is discarded, and the recursive backtracking approach continues.

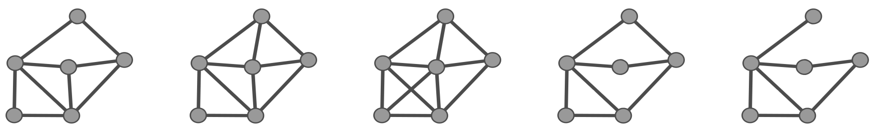
4 Results

The results in this section were generated by experiments performed on a Linux server with a 2 GHz processor, limited to 5GB of physical memory.

4.1 Network Data

The SocioPatterns project was set up to study patterns in social dynamics [6]. The SocioPatterns sensing platform gathers face-to-face proximity data from participants who wear wireless sensors. The data forms a contact network, in which a node represents a person and an interaction represents face-to-face contact. The number of interactions between two people represents the number of times they came into contact. The data is a useful proxy for communication.

One such experiment was performed at the ACM Conference on Hypertext and Hypermedia 2009 in Turin, Italy. The Hypertext 2009 network contains 113 nodes and 9 865 interactions. Another experiment was performed at the Science Gallery in Dublin during the INFECTIOUS: STAY AWAY exhibition in 2009. The Infectious network contains 410 nodes and 17 298 interactions. The d-value we chose for the Hypertext 2009 and Infectious data sets was 10 minutes, to reflect the contact dynamics.



(a) Query. (b) One more interaction. (c) Two more interactions. (d) One less interaction. (e) Two less interactions.

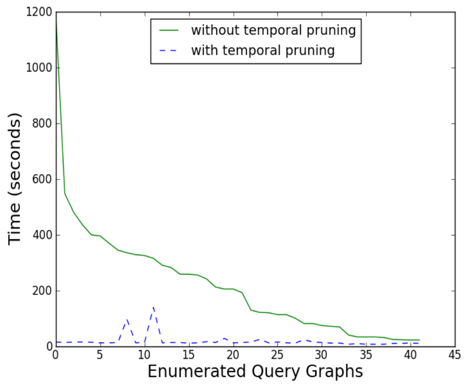
Fig. 6: A sample query graph and its topological variations. (6a) The original query graph. (6b, 6c) Embeddings which match inexactly, with one or two more interactions than the query graph. (6d, 6e) Embeddings with one or two fewer interactions than specified by the query graph.

We selected 42 small connected graphs to act as query graphs in the matching process (see an example and some variations in Fig. 6). Half of these have five nodes, and the other half have six. The diameter of the query graphs ranges from one to four. The diameter of a graph is the longest of all shortest paths between any two nodes in the graph.

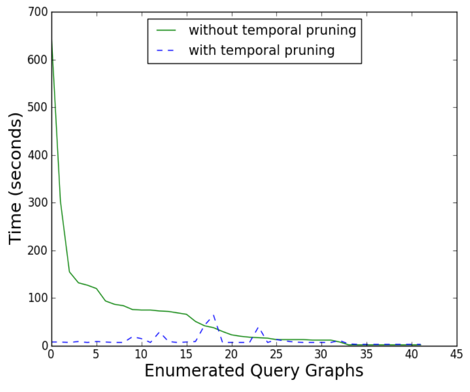
4.2 Inexact Matches

We ran three experiments on each data set. These allowed a cost threshold θ of zero, one and two. The threshold restricts the number of interactions in the embedding that are allowed to vary from the query graph. When θ is zero, the match is exact. When θ is one, an interaction may be missing from the embedding or an extra interaction my be present. When θ is two, either two interactions are missing or two extra interactions are in the embedding. Examples of some permissible variations to a query graph are shown in Fig. 6.

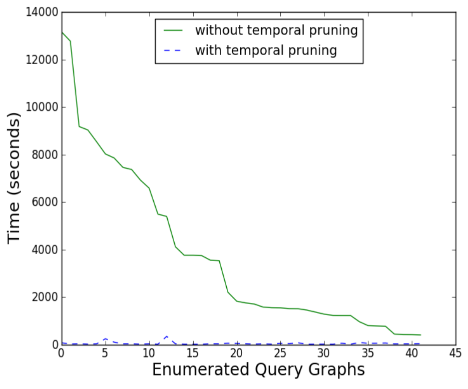
Although we allow the structure of embedded subgraphs to vary to a certain extent, we do not allow the temporal constraint to be violated. Thus, the examples shown in Fig. 6 must be temporally constrained, for any instance of their embedding. This ensures that the embedded structures will be interpretable in a temporal context. The temporal constraints can be relaxed by increasing the time scale over which interactions take place, by changing the d-value.



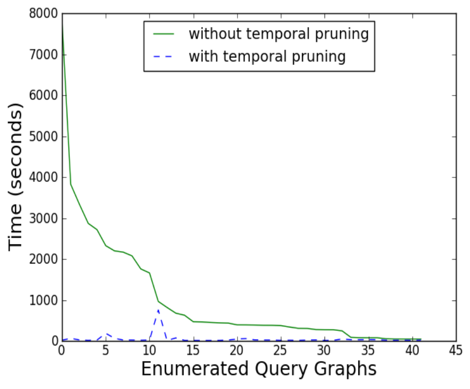
(a) Hypertext - exact matching.



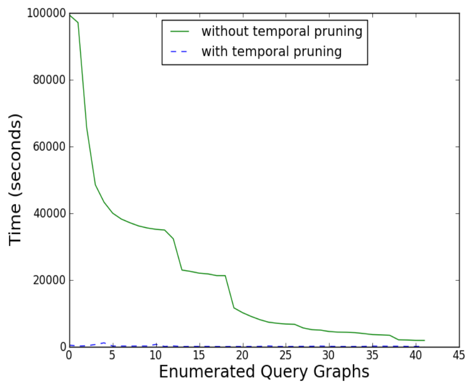
(b) Infectious - exact matching.



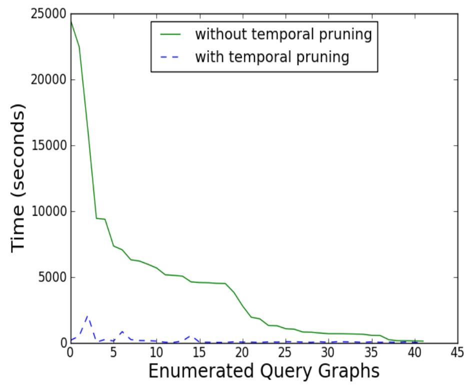
(c) Hypertext - one interaction error allowed.



(d) Infectious - one interaction error allowed.



(e) Hypertext - two errors allowed.



(f) Infectious - two interaction errors allowed.

Fig. 7: The time taken for all embeddings of each query graph to be found in the data sets, with a cost thresholds of $\theta = \{0, 1, 2\}$. Each plot compares the time taken for the process with and without the use of temporal information. The x-axis lists the query graphs sought, in decreasing order from the one that took longest to find without temporal pruning. The biggest peaks for the temporal pruning approach occur for query graphs with cycles, which have many embeddings in the network, leading to a slower matching process.

4.3 Processing Time

To show that temporal information can speed up the matching process, we recorded the time taken for embeddings of our set of query graphs to be found in the Hypertext 2009 and Infectious data sets. The results are plotted in Fig. 7. In all cases, independent of the error threshold, the query graphs are found faster when temporal information is used. In many cases, the difference is notable .

It appears that temporal pruning always allows the search to terminate faster. To see why, consider the following scenario. From a state s in a partial match, the next candidate node pair (a node from $G1$ and one from $G2$) is evaluated. If the pair is to be included in the match, the topology of the subgraphs they induce (in $G1$ and $G2$ respectively) must match. If the match is correct, the search will continue. However, if temporal information is used, then the embedding in $G1$ must be temporally constrained. This facilitates an earlier discarding of the match. The result is that fewer embeddings are returned, with each embedding being temporally constrained.

Given that the number of interactions present in the Infectious network is approximately twice that of those present in the Hypertext 2009 network, (17 298 versus 9 865 respectively), it is interesting to see that the temporal pruning approach is still almost always faster than the static approach.

In the case where no temporal pruning is used, there is a pronounced difference in the time taken for embeddings of query graphs to be identified. This is due to the fact that queries with longer diameters – composed of longer paths – take longest to find without temporal pruning. Those which are found quickest have short diameters, for example cliques and near-cliques. This may be extremely useful in real-world scenarios, when paths of contacts occurring close in time are present.

5 Conclusion

When errors are present in network data, or when a user wants to find a range of subgraphs similar to the one they consider important, inexact subgraph matching comes into play. The solution to this problem is more computationally expensive than exact matching. Traditionally, labels on the network nodes are used to constrain the search. However, in many temporal networks, such node labels are not available. Thus, we have introduced temporal inexact subgraph matching, using temporal information to prune the search space. The returned embeddings have the property of being temporally constrained, such that interactions in the network take place within a given time window of each other.

When applied to two undirected contact networks, our approach outperformed the corresponding static inexact subgraph matching algorithm in terms of processing time. From among the query graphs we sought, the greatest benefit of the temporal method occurred when the diameter of the query graph was longer. This may be useful when seeking longer paths in networks, for example to find disease spreading in epidemic networks, or information diffusion in communication networks.

References

1. Baños, R.A., Borge-Holthoefer, J., Moreno, Y.: The role of hidden influentials in the diffusion of online information cascades. CoRR (2013), `http://arxiv.org/abs/1303.4629`
2. Bearman, P., Moody, J., Stovel, K.: Chains of affection: The structure of adolescent romantic and sexual networks. American J. of Sociology 110, 44–91 (2004)
3. Cordella,L. P., Foggia,P., Sansone, C., Vento, M.: An Improved Algorithm for Matching Large Graphs. 3^{rd} IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition pp. 149–159 (2001)
4. Hofer, H., Borgelt, C., Berthold, M.R.: Large scale mining of molecular fragments with wildcards. Intelligent Data Analysis 8(5) (2003)
5. Holme, P., Saramäki, J.: Temporal networks. CoRR abs/1108.1780 (2011), `http://arxiv.org/abs/1108.1780`
6. Isella, L., Stehle, J., Barrat, A., Cattuto, C., Pinton, J., Broeck, W.: What's in a crowd? analysis of face-to-face behavioral networks. Journal of Theoretical Biology 271(1), 166–180 (2011)
7. Kempe, D., Kleinberg, J., Kumar, A.: Connectivity and inference problems for temporal networks. Journal of Computer and System Sciences 76(036113) (2002)
8. Kivelä, M., Pan, R.K., Kaski, K., Kertész, J., Saramäki, J., Karsai, M.: Multiscale analysis of spreading in a large communication network. CoRR (2011), `http://arxiv.org/abs/1112.4312`
9. Leskovec, J., Singh, A., Kleinberg, J.: Patterns of influence in a recommendation network. Advances in Knowledge Discovery and Data Mining pp. 380–389 (2006)
10. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: simple building blocks of complex networks. Science 298(5594), 824–827 (2002)
11. Moody, J.: The importance of relationship timing for diffusion. Social Forces 81, 25–56 (2002)
12. NetworkX Developers: NetworkX. networkx.lanl.gov (2010)
13. Redmond, U., Cunningham, P.: A temporal network analysis reveals the unprofitability of arbitrage in The Prosper Marketplace. Expert Systems with Applications 40(9), 3715–3721 (7 2013), `http://www.sciencedirect.com/science/article/pii/S0957417412013188`
14. Tian, Y., Patel, J.M.: Tale: A tool for approximate large graph matching. 2008 IEEE 24th International Conference on Data Engineering pp. 963–972 (2008)
15. Tong, H., Gallagher, B., Faloutsos, C., Eliassi-Rad, T.: Fast best-effort pattern matching in large attributed graphs. Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07 (2007)
16. Tsai, W., Fu, K.: Error-correcting isomorphisms of attributed relational graphs for pattern analysis. IEEE Trans. on Systems, Man and Cybernetics 9(12) (1979)
17. Tsai, W., Fu, K.: Subgraph error-correcting isomorphisms for syntactic pattern recognition. IEEE Trans. on Systems, Man, and Cybernetics SMC-13, 48–62 (1983)
18. Wang, J., Zhang, K., Chirn, G.: The approximate graph matching problem. Proc. of the 12th IAPR Intl Conf. on Pattern Recognition 2, 284–288 (1994)
19. Zhao, Q., Tian, Y., He, Q., Oliver, N., Jin, R., Lee, W.C.: Communication motifs: A tool to characterize social communications. In Proceedings of the 19th ACM International Conference on Information and Knowledge Management p. 1645 (2010)

Prequential AUC for Classifier Evaluation and Drift Detection in Evolving Data Streams

Dariusz Brzezinski and Jerzy Stefanowski

Institute of Computing Science, Poznan University of Technology,
ul. Piotrowo 2, 60–965 Poznan, Poland
{dariusz.brzezinski,jerzy.stefanowski}@cs.put.poznan.pl

Abstract. Detecting and adapting to concept drift makes learning data stream classifiers a difficult task. It becomes even more complex when the distribution of classes in the stream becomes imbalanced. Currently, proper assessment of classifiers for such data is still a challenge, as existing evaluation measures either do not take into account class imbalance or are unable to indicate class ratio changes in time. In this paper, we advocate the use of the area under the ROC curve (AUC) in imbalanced data stream settings and propose an incremental algorithm that uses a sorted tree structure with a sliding window to compute AUC using constant time and memory. Additionally, we experimentally verify that this algorithm is capable of correctly evaluating classifiers on imbalanced streams and can be used as a basis for detecting sudden changes in class definitions and imbalance ratio.

Keywords: AUC, data stream, class imbalance, concept drift

1 Introduction

Many modern information system, e.g. concerning sensor networks, recommender systems, or traffic monitoring, record and process huge amounts of data. However, the massive size and complexity of the collected datasets make the discovery of patterns hidden in the data a difficult task. Such limitations are particularly visible when mining data in the form of transient *data streams*. Stream processing imposes hard requirements concerning limited amount of memory and small processing time, as well as the need of reacting to *concept drifts*, i.e., changes in distributions and definitions of target classes over time. For supervised classification, these requirements mean that newly proposed classifiers should not only accurately predict class labels of incoming examples, but also adapt to concept drifts while satisfying computational restrictions.

Classification becomes even more difficult if the data complexities also include *class imbalance*. It is an obstacle even for learning from static data, as classifiers are biased toward the majority classes and tend to misclassify minority class examples. However, it has been also shown that class imbalance ratio is usually not the only factor that impedes learning. Experimental studies [1, 2] suggest that when additional data complexities occur together with class imbalance, the deterioration of classification performance is amplified and affects mostly the minority class. In this paper, we focus our attention on the complexity resulting from the combination of class imbalance, stream processing, and concept-drift.

Although for static imbalanced data several specialized learning techniques have recently been introduced [3, 4], similar research in the context of data streams is limited to a few papers [5–8]. However, these studies show that evolving and imbalanced data streams are particularly demanding learning scenarios, and the problem of effectively evaluating a classifier is vitally important for such data.

Currently, the performance of data stream classifiers is commonly measured with predictive accuracy (or respective error), which is usually calculated in a cumulative way over all incoming examples or at selected points in time when examples are processed in blocks. However, when values of these measures are averaged over an entire stream, they loose information about the classifier's reactions to drifts. Even recent proposals including a *prequential* way of calculating accuracy [9] or using the Kappa statistic [10, 11] are not sufficient as they are unable to depict changes in class distribution, which could appear in different moments of evolving data streams. Moreover, when the ratio of positive to negative instances changes in a test set, a classifier chosen using these metrics may no longer perform sufficiently good, or even acceptably [12].

For static imbalanced problems, a popular alternative to accuracy is the area under the ROC (Receiver Operator Characteristic) curve (AUC). An important property of AUC is that it is invariant to changes in class distribution. Moreover, for scoring classifiers it has a very useful statistical interpretation as the expectation that a randomly drawn positive example receives a higher score than a random negative example. Thus, it measures the ranking ability of classifiers, which is especially desirable if one wants to dynamically change the classification threshold in response to changing class or cost distributions [12]. Finally, several authors have shown that AUC is more preferable for model evaluation than total accuracy [13].

However, in order to calculate AUC, one needs to sort a given dataset and iterate through each example. Because the sorted order of examples defines the resulting value, adding an example to the dataset forces the procedure to be repeated. Therefore, AUC cannot be directly computed on data streams, as this would require $O(n)$ time and memory at each time point, where n is the current length of the data stream (if previously sorted scores are preserved, one only needs to insert a new score and linearly scan through the examples to calculate AUC). Up till now, the use of AUC for data streams has been limited to estimations on periodical holdout sets [8, 6] or entire streams [5, 7], making it either potentially biased or computationally infeasible.

In this paper, we propose a new approach for calculating AUC incrementally with limited time and memory requirements. The proposed algorithm incorporates a sorted tree structure with a sliding window as a forgetting mechanism, making it both computationally feasible and appropriate for concept-drifting streams. According to our best knowledge, such an approach has not been considered in the literature. Furthermore, we argue that, compared to standard accuracy, the analysis of changes of prequential AUC over time could provide more information about the performance of classifiers with respect to different types of drifts, in particular for streams with evolving class imbalance ratio. To verify this hypothesis, we carry out experiments with several synthetic and real datasets representing scenarios involving different types of drift, including sudden changes in the class imbalance ratio.

The remainder of the paper is organized as follows. Section 2 presents related work. In Section 3, we introduce an algorithm for calculating prequential AUC and investigate its properties, while Section 4 shows how prequential AUC can be used for concept drift detection. In Section 5, we present experimental results on real and synthetic datasets, which demonstrate the properties of the proposed algorithms. Finally, in Section 6 we draw conclusions and discuss future research.

2 Evaluating Data Stream Classifiers

In data stream mining, predictive abilities of a classifier are evaluated by using a holdout test set, chunks of examples, or incrementally after each example [14]. More recently, Gama et al. [9] proposed prequential accuracy with forgetting as a means of evaluating data stream classifiers and enhancing drift detection methods. They have shown that computing accuracy only over the most recent examples, instead of the entire stream, is more appropriate for continuous assessment and drift detection in evolving data streams. Nevertheless, prequential accuracy inherits the weaknesses of traditional accuracy, that is, variance with respect to class distribution and promoting majority class predictions.

For imbalanced data streams, Bifet and Frank [10] proposed the use of the Kappa statistic with a sliding window. Furthermore, this metric has been recently extended to take into account temporal dependence [11]. However, the Kappa statistic requires a baseline classifier, which is dependent of the current class imbalance ratio. Furthermore, in contrast to accuracy, the Kappa statistic is a relative measure without a probabilistic interpretation, meaning that its value alone does not directly state whether a classifier will predict accurately enough in a given setting, only that it performs better than general baselines.

The AUC measure has also been used for imbalanced data streams, however, in a limited way. Some researchers chose to calculate AUC using entire streams [5, 7], while others used periodical holdout sets [8, 6]. Nevertheless, it was noticed that periodical holdout sets may not fully capture the temporal dimension of the data, whereas evaluation using entire streams is neither feasible for large datasets nor suitable for drift detection. It is also worth mentioning that an algorithm for computing AUC incrementally has also been proposed [15], yet one which calculates AUC from all available examples and is not applicable to evolving data streams. Although the cited works show that AUC is recognized as a measure which should be used to evaluate classifiers for imbalanced data streams, up till now it has been computed the same way as for static data. In the following sections, we propose a simple and efficient algorithm for calculating

AUC incrementally with forgetting, and investigate its properties with respect to classifier evaluation and drift detection in evolving data streams.

3 Prequential AUC

Our main interest in this paper is to evaluate data stream classifiers for evolving imbalanced data streams. For this purpose, we advocate the use of the area under the receiver operator characteristic curve (AUC). Therefore, we will consider scoring classifiers, i.e., classifiers that for each predicted class label additionally return a numeric value (score) indicating the extent to which an instance is predicted to be positive or negative. Furthermore, we will limit our analysis to binary classification. It is worth mentioning, that most classifiers can produce scores, and those that only predict class labels can be converted to scoring classifiers [12].

We propose to compute AUC incrementally using a forgetting mechanism that employs a sorted window of classification scores of the most recent examples. It is worth noting that, since the calculation of AUC requires sorting examples with respect to their classification scores, it cannot be computed on an entire stream or using fading factors without using additional memory. To efficiently maintain a sorted set of scores, we propose to use a *red-black tree* [16], which is capable of adding and removing elements in logarithmic time without any additional memory. Furthermore, a window of scores is required to identify the age of each score. With these two structures, for each incoming example a new score is inserted into the window (line 16) as well as the tree (line 11) and, if the window of examples has been exceeded, the oldest score is removed (lines 5 and 16). After the window has been updated, AUC is calculated by summing the number of positive examples occurring before each negative example (lines 20–24) and normalizing that value by all possible pairs pn (line 25), where p is the number of positives and n is the number of negatives in the window. This method of calculating AUC is equivalent to summing the area of trapezoids for each pair of sequential points in the ROC curve [12], but is more suitable for our purposes as it requires very little computation given a sorted collection of scores. Algorithm 1 lists the pseudo-code for calculating prequential AUC.

Let us now analyze the complexity of the proposed approach. For a window of size d, the time complexity of adding and removing a score to the red-black tree is $O(2 \log d)$. Additionally, the computation of AUC requires iterating through all the scores in the tree, which is an $O(d)$ operation. In summary, the computation of prequential AUC has a complexity of $O(d + 2 \log d)$ per example and since d is a user-defined constant this resolves to a complexity of $O(1)$. It is worth noticing that if AUC only needs to be sampled every k examples (a common scenario while plotting metrics in time) lines from 19 to 25 can be executed only once per k examples. In terms of space complexity, the algorithm requires $O(2d)$ memory for the red-black tree and window, which also resolves to $O(1)$.

In contrast to error-rate performance metrics, such as accuracy [9, 14] or the Kappa statistic [10, 11], the proposed measure is invariant of the class distribu-

Algorithm 1 Prequential AUC

Input: $\mathcal{S}$: stream of examples, d: window size
Output: $\hat{\theta}$: prequential AUC after each example

```
 1: $W \leftarrow \emptyset$; $n \leftarrow 0$; $p \leftarrow 0$; $idx \leftarrow 0$;
 2: for all scored examples $\mathbf{x}^t \in \mathcal{S}$ do
 3:    // Remove oldest score from the window
 4:    if $idx \geq d$ then
 5:       $scoreTree.remove(W[idx \textbf{ mod } d])$;
 6:       if $isPositive(W[idx \textbf{ mod } d])$ then
 7:          $p \leftarrow p - 1$;
 8:       else
 9:          $n \leftarrow n - 1$;
10:    // Add new score to the window
11:    $scoreTree.add(\mathbf{x}^t)$;
12:    if $isPositive(\mathbf{x}^t)$ then
13:       $p \leftarrow p + 1$;
14:    else
15:       $n \leftarrow n + 1$;
16:    $W[idx \textbf{ mod } d] \leftarrow \mathbf{x}^t$;
17:    $idx \leftarrow idx + 1$;
18:    // Calculate AUC
19:    $AUC \leftarrow 0$; $c \leftarrow 0$;
20:    for all consecutive scored examples $s \in scoreTree$ do
21:       if $isPositive(s)$ then
22:          $c \leftarrow c + 1$;
23:       else
24:          $AUC \leftarrow AUC + c$;
25:    $\hat{\theta} \leftarrow \frac{AUC}{pn}$;
```

tion. Furthermore, unlike accuracy it does not promote majority class predictions. Additionally, in contrast to the Kappa statistic, AUC is a non-relative, $[0, 1]$ normalized metric with a direct statistical interpretation. As opposed to previous applications of AUC to data streams [5–8], the proposed algorithm can be executed after each example using constant time and memory. Finally, compared to the method presented in [15], the proposed algorithm provides a forgetting mechanism and uses a sorting structure, making it suitable for evolving data streams and allowing for efficient sampling.

4 Drift Detection Using AUC

Prequential AUC assesses the ranking abilities of a classifier and is invariant of the class distribution. These properties differentiate it from common evaluation metrics for data stream classifiers and could be applied in an additional context. In particular, for streams with high class imbalance ratios simple metrics, such as accuracy, will suggest good performance (as they are biased toward recognizing the majority class) and may poorly exhibit concept drifts. Therefore, we propose to investigate AUC not only as an evaluation measure, but also as basis for drift detection in imbalanced streams, where it should better indicate changes concerning the minority class.

For this purpose, we propose to modify the Page-Hinkley (PH) test [9], however, generally other drift detection methods could also have been adapted. The PH test considers a variable m^t, which measures the accumulated difference between the observed values e (originally error estimates) and their mean till the current moment, decreased by a user-defined magnitude of allowed changes δ: $m^t = \sum_{i=1}^{t} (e^t - \bar{e}^t - \delta)$. After each observation e^t, the test checks whether the difference between the current m^t and the smallest value up to this moment $\min(m^i, i = 1, \ldots, t)$ is greater than a given threshold λ. If the difference exceeds λ, a drift is signaled. In this paper, we propose to use the area *over* the ROC curve $(1 - AUC)$ as the observed value. Hence, according to the statistical interpretation of AUC, instead of error estimates, we monitor the estimate of the probability that a randomly chosen positive is ranked *after* a randomly chosen negative. This way, the PH test will trigger whenever a classifier begins to make severe ranking errors regardless of the class imbalance ratio.

The aim of using prequential AUC as an evaluation measure is to provide accurate classifier assessment and drift detection for evolving imbalanced streams. In the following section, we examine the characteristics of the proposed metric in scenarios involving different types of drifts and imbalance ratios.

5 Experiments

We performed two groups of experiments, one showcasing the properties of prequential AUC as an evaluation metric, and another assessing its effectiveness as a basis for drift detection. In the first group, we tested five different classifiers [14, 17]: Naive Bayes (NB), Very Fast Decision Tree with Naive Bayes leaves (VFDT), Dynamic Weighted Majority (DWM), Online Bagging with an ADWIN drift detector (Bag), and Online Accuracy Updated Ensemble (OAUE). Naive Bayes and VFDT were chosen as incremental algorithms without any forgetting mechanism, Online Bagging was chosen as an algorithm with a drift detector, while OAUE and DWM were selected as representatives of ensemble learners. For the second group of experiments, we only utilized VFDT with Naive Bayes leaves, similarly as was done in [9].

All the algorithms and evaluation methods were implemented in Java as part of the MOA framework [18]. The experiments were conducted on a machine equipped with a dual-core Intel i7-2640M CPU, 2.8Ghz processor and 16 GB of RAM. For all the ensemble methods (Bag, DWM, OAUE) we used 10 Very Fast Decision Trees as base learners, each with a grace period $n_{min} = 100$, split confidence $\delta = 0.01$, and tie-threshold $\tau = 0.05$ [14].

5.1 Datasets

For the first group of experiments, with prequential AUC as an evaluation metric, we used 2 real and 10 synthetic datasets[1]. The real datasets were Airlines (`Air`) and PAKDD'09 (`PAKDD`), representing a balanced and imbalanced dataset respectively. To create synthetic datasets we used three popular data stream generators from MOA: SEA (`SEA`), Hyperplane (`Hyp`), and Random RBF (`RBF`) [18]. More precisely, `SEA` was a dataset without drift, $\texttt{SEA}_x$ were datasets with a 1:x class ratio and three sudden drifts, and $\texttt{SEA}_{RC}$ contained three class ratio changes (1:1 $\rightarrow$ 1:100 $\rightarrow$ 1:10 $\rightarrow$ 1:1). Furthermore, `RBF` contained two very short changes (blips), whereas $\texttt{Hyp}_x$ were datasets with a 1:x class ratio and a slow incremental drift throughout the entire stream.

For assessing prequential AUC as a measure for monitoring drift, we created 7 synthetic datasets using the SEA (`SEA`), RBF (`RBF`), Random Tree (`RT`), and Agrawal (`Agr`) generators [18]. Each dataset tested for a single reaction (or lack of one): $\texttt{SEA}_{NoDrift}$ contained no changes, and should not trigger any drift detector; `RT` involved a sudden change after 30 k examples; $\texttt{Agr}_1$, $\texttt{Agr}_{10}$, $\texttt{Agr}_{100}$ also contained a sudden change after 30 k examples, but had a 1:1, 1:10, 1:100 class ratio, respectively; $\texttt{SEA}_{Ratio}$ included a sudden 1:1 $\rightarrow$ 1:100 ratio change after 10 k examples; $\texttt{RBF}_{Blips}$ contained two blips, which should not trigger the detector. The main characteristics of all the datasets are given in Table 1.

Table 1. Characteristic of datasets.

Dataset	#Inst	#Attrs	Class ratio	Noise	#Drifts	Drift type
`SEA`	100 k	3	1:1	10%	3	none
$\texttt{SEA}_x$	1 M	3	1:x	10%	3	sudden
$\texttt{Hyp}_x$	500 k	5	1:x	5%	1	incremental
`RBF`	1 M	20	1:1	0%	2	blips
$\texttt{SEA}_{RC}$	1 M	3	1:1/1:100/1:10	10%	4	virtual
`Air`	539 k	7	1:1	-	-	unknown
`PAKDD`	50 k	30	1:4	-	-	unknown
$\texttt{SEA}_{NoDrift}$	20 k	3	1:1	10%	1	none
$\texttt{Agr}_x$	40 k	9	1:x	1%	1	sudden
`RT`	40 k	10	1:1	0%	1	sudden
$\texttt{SEA}_{Ratio}$	40 k	3	1:1/1:100	10%	1	virtual
$\texttt{RBF}_{Blips}$	40 k	20	1:1	0%	2	blips

5.2 Results

All of the analyzed algorithms were tested in terms of accuracy and prequential AUC. In the first group of experiments, the results were obtained using the

[1] Source code, test scripts, generator parameters, and links to datasets available at: `http://www.cs.put.poznan.pl/dbrzezinski/software.php`

test-then-train procedure [14], with a sliding window of 1000 examples. Table 2 presents a comparison of average classification accuracy and prequential AUC.

Table 2. Average prequential accuracy (Acc.) and AUC (AUC).

	NB		VFDT		Bag		DWM		OAUE	
	Acc.	AUC	Acc.	AUC	Acc.	AUC	Acc.	AUC	Acc.	AUC
SEA$_{ND}$	0.86	0.90	0.89	0.89	0.89	0.90	0.89	0.90	0.89	0.90
SEA$_1$	0.84	0.88	0.85	0.87	0.89	0.88	0.89	0.88	0.89	0.88
SEA$_{10}$	0.84	0.74	0.87	0.73	0.89	0.74	0.89	0.74	0.89	0.74
SEA$_{100}$	0.89	0.54	0.89	0.54	0.90	0.54	0.90	0.54	0.90	0.54
Hyp$_1$	0.78	0.85	0.81	0.87	0.88	0.93	0.88	0.92	0.88	0.93
Hyp$_{10}$	0.88	0.80	0.89	0.74	0.91	0.81	0.91	0.76	0.91	0.82
Hyp$_{100}$	0.94	0.57	0.93	0.53	0.94	0.56	0.94	0.52	0.94	0.55
RBF	0.74	0.83	0.96	0.98	0.99	1.00	0.98	1.00	0.99	1.00
SEA$_{RC}$	0.86	0.77	0.89	0.77	0.90	0.77	0.89	0.77	0.90	0.77
Air	0.65	0.66	0.64	0.65	0.64	0.65	0.65	0.65	0.67	0.68
PAKKD	0.56	0.64	0.73	0.57	0.80	0.63	0.80	0.50	0.80	0.62

By comparing average values of the analyzed evaluation metrics, we can see that for datasets with a balanced class ratio (SEA, SEA$_1$, Hyp$_1$, RBF, Air) both measures have similar values. As we expected, for datasets with class imbalance (SEA$_{10}$, SEA$_{100}$, Hyp$_{10}$, Hyp$_{100}$, PAKKD, SEA$_{RC}$) accuracy does not demonstrate the difficulties the classifiers have with recognizing minority class examples. The differences between accuracy and AUC are even more visible on graphical plots depicting algorithm performance in time. Figures 1–5 present selected performance plots, which best characterize the differences between both metrics.

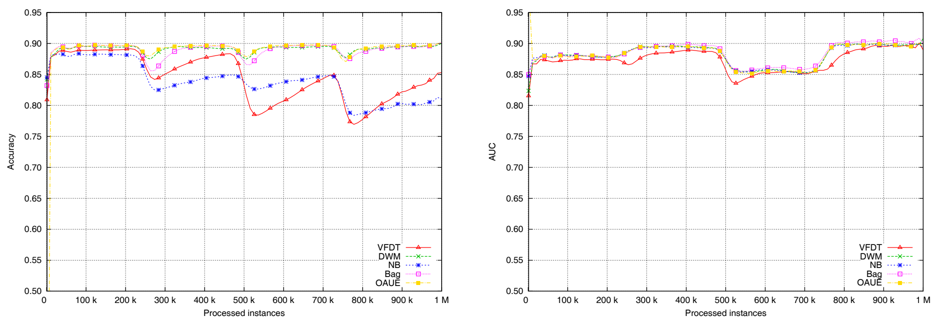


Fig. 1. Prequential accuracy (left) and AUC (right) on a data stream with sudden drifts and a balanced class ratio.

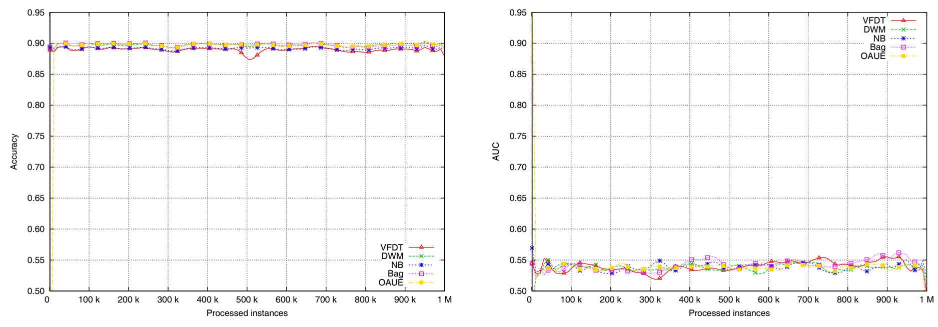


Fig. 2. Prequential accuracy (left) and AUC (right) on a data stream with sudden drifts and 1:100 class imbalance ratio.

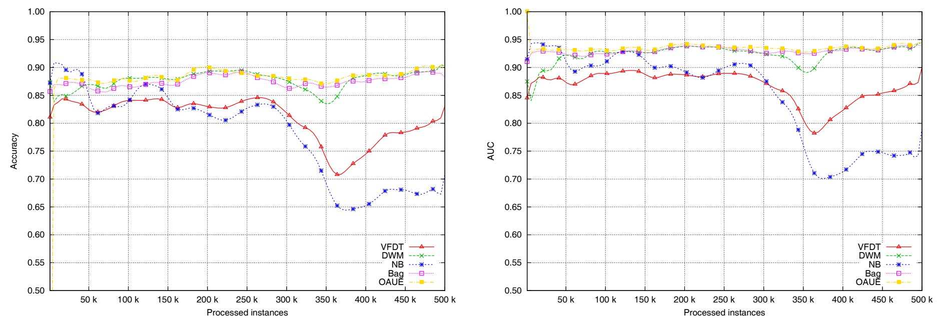


Fig. 3. Prequential accuracy (left) and AUC (right) on a data stream with incremental drift and a balanced class ratio.

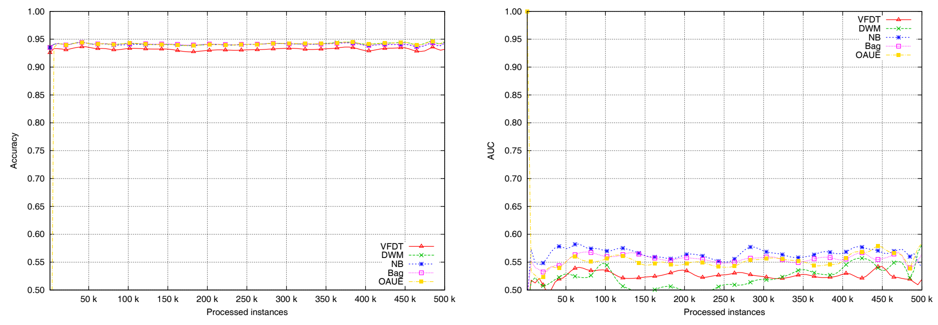


Fig. 4. Prequential accuracy (left) and AUC (right) on a data stream with incremental drift and 1:100 class imbalance ratio.

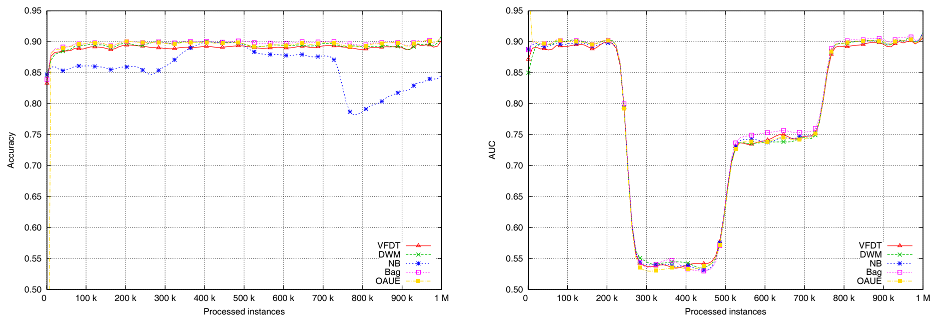


Fig. 5. Prequential accuracy (left) and AUC (right) for data with class ratio changes.

Comparing Figures 1 and 2, we can notice how the class imbalance ratio affects both prequential accuracy and AUC. The accuracy plot visibly flattens when class imbalance rises, but absolute values almost do not change. AUC on the other hand flattens but its value drastically changes, showing more clearly the classifiers' inability to recognize the minority class.

A similar situation is visible on Figures 3 and 4, where the classifiers were subject to an ongoing slow incremental drift. When classes are balanced, the plots are almost identical, both in terms of shape and absolute values. However, when the class ratio is equal 1:100, the accuracy plot flattens and its average value rises, while the AUC plot still clearly shows that classifiers are unstable and additionally its average value signals poor performance.

Finally, Figure 5 depicts classifier performance for a data stream with class ratio changes, which are sometimes called virtual drift. Apart from NB, all the tested classifiers kept the same accuracy after each drift making the changes invisible on the performance plot. However, on the AUC plot, ratio changes are clearly visible providing valuable information about the ongoing processes in the stream. In fact, the absolute values of AUC hint the severity of class imbalance in a given moment in time. This situation illustrates the advantages of prequential AUC as a measure for indicating class ratio changes.

The second group of experiments involved using the PH test to detect drifts based on changes in prequential accuracy and AUC. To compare both metrics, we used window sizes (1000–5000) and test parameters $\lambda = 100$, $\delta = 0.1$, as proposed in [9]. Table 3 presents the number of missed versus false detection counts, with average delay time for correct detections. The results refer to total counts and means over 10 runs of streams generated with different seeds.

Concerning datasets with balanced classes, both evaluation metrics provide similar drift detection rates and delays. However, for datasets with high class imbalance the PH test notes more missed detections for accuracy. This is probably due to the plot "flattening" caused by promoting majority class predictions. On the other hand, detectors which use AUC have less missed detections for highly imbalanced streams, but still suffer from a relatively high number of false alarms. This suggests that detectors using AUC should probably be parametrized dif-

Table 3. Number of missed and false detections (in the format missed:false) obtained using the PH test with accuracy (Acc) and AUC (AUC). Average delays of correct detections are given in parenthesis, where (-) means that the detector was not triggered or datasets did not contain any change. Subscripts in column names indicate the number of examples used for estimating errors.

	Acc_{1k}	Acc_{2k}	Acc_{3k}	Acc_{4k}	Acc_{5k}
$\texttt{SEA}_{NoDrift}$	0:0 (-)	0:0 (-)	0:0 (-)	0:0 (-)	0:0 (-)
$\texttt{Agr}_{1}$	0:2 (1040)	0:1 (1859)	0:0 (2843)	1:0 (4033)	5:0 (4603)
$\texttt{Agr}_{10}$	0:9 (1202)	0:3 (1228)	0:2 (1679)	0:2 (2190)	0:2 (2817)
$\texttt{Agr}_{100}$	2:12 (1610)	2:17 (2913)	2:10 (3136)	3:12 (3903)	3:10 (4612)
RT	6:0 (1843)	7:0 (2621)	8:0 (2933)	8:0 (3754)	8:0 (4695)
$\texttt{SEA}_{Ratio}$	10:0 (-)	10:0 (-)	10:0 (-)	10:0 (-)	10:0 (-)
$\texttt{RBF}_{Blips}$	0:2 (-)	0:1 (-)	0:0 (-)	0:0 (-)	0:0 (-)
	AUC_{1k}	AUC_{2k}	AUC_{3k}	AUC_{4k}	AUC_{5k}
$\texttt{SEA}_{NoDrift}$	0:0 (-)	0:0 (-)	0:0 (-)	0:0 (-)	0:0 (-)
$\texttt{Agr}_{1}$	2:2 (1042)	3:1 (1760)	4:1 (2726)	4:0 (3773)	7:0 (4640)
$\texttt{Agr}_{10}$	0:5 (868)	0:5 (1539)	0:1 (1506)	0:1 (1778)	1:1 (2197)
$\texttt{Agr}_{100}$	0:19 (1548)	0:18 (2461)	1:9 (2664)	1:11 (3563)	2:9 (4835)
RT	3:0 (1815)	5:0 (2407)	6:0 (3105)	6:0 (4121)	7:0 (4725)
$\texttt{SEA}_{Ratio}$	0:0 (1339)	0:0 (2249)	0:0 (3152)	0:0 (4057)	0:0 (4959)
$\texttt{RBF}_{Blips}$	0:3 (-)	0:1 (-)	0:0 (-)	0:0 (-)	0:0 (-)

ferently than those using accuracy. However, the most visible differences are for streams with class ratio changes. The PH test misses all virtual drifts when using accuracy as the base metric, but detects all the drifts when prequential AUC is used. This shows, that in imbalanced evolving environments the use of AUC as an evaluation measure could be of more value than standard accuracy.

6 Conclusions

In case of static data, AUC is a useful metric for evaluating classifiers both on balanced and imbalanced classes. However, up till now it has not been sufficiently popular in data stream mining, due to its costly calculation. In this paper, we introduced an efficient method for calculating AUC incrementally with forgetting on evolving data streams. The proposed algorithm, called prequential AUC, proved to be useful for visualizing classifier performance over time and as a basis for drift detection. In particular, experiments involving real and synthetic datasets have shown that prequential AUC is capable of correctly identifying poor classifier performance on imbalanced streams and detecting virtual drifts, i.e., changes in class ratio over time.

As our ongoing research, we are analyzing the possibility of using variations of AUC, such as scored AUC [12], to detect drifts more rapidly. Furthermore, we plan to analyze ROC curves plotted over time as a means of in-depth assessment of classifier performance on evolving data streams.

Acknowledgments. The authors' research was partially funded by the Polish National Science Center under Grant No. DEC-2013/11/B/ST6/00963.

References

1. Batista, G., Prati, R.C., Monard, M.C.: A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explorations Newsletter **6(1)** (2004) 20–29
2. Japkowicz, N., Stephen, S.: The class imbalance problem: A systematic study. Intell. Data Anal. **6**(5) (2002) 429–449
3. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Trans. Knowl. Data Eng. **21**(9) (2009) 1263–1284
4. He, H., Ma, Y., eds.: Imbalanced Learning: Foundations, Algorithms, and Applications. Wiley-IEEE Press (2013)
5. Ditzler, G., Polikar, R.: Incremental learning of concept drift from streaming imbalanced data. IEEE Trans. Knowl. Data Eng. **25**(10) (2013) 2283–2301
6. Hoens, T.R., Chawla, N.V.: Learning in non-stationary environments with class imbalance. In: KDD, ACM (2012) 168–176
7. Lichtenwalter, R., Chawla, N.V.: Adaptive methods for classification in arbitrarily imbalanced and drifting data streams. In: PAKDD Workshops. Volume 5669 of Lecture Notes in Computer Science., Springer (2009) 53–75
8. Wang, B., Pineau, J.: Online ensemble learning for imbalanced data streams. CoRR **abs/1310.8004** (2013)
9. Gama, J., Sebastião, R., Rodrigues, P.P.: On evaluating stream learning algorithms. Machine Learning **90**(3) (2013) 317–346
10. Bifet, A., Frank, E.: Sentiment knowledge discovery in twitter streaming data. In: Discovery Science. Volume 6332 of Lecture Notes in Computer Science., Springer (2010) 1–15
11. Zliobaite, I., Bifet, A., Read, J., Pfahringer, B., Holmes, G.: Evaluation methods and decision theory for classification of streaming data with temporal dependence. Machine Learning (2014)
12. Wu, S., Flach, P.A., Ramirez, C.F.: An improved model selection heuristic for AUC. In: ECML. Volume 4701 of Lecture Notes in Computer Science., Springer (2007) 478–489
13. Huang, J., Ling, C.X.: Using AUC and accuracy in evaluating learning algorithms. IEEE Trans. Knowl. Data Eng. **17**(3) (2005) 299–310
14. Gama, J.: Knowledge Discovery from Data Streams. Chapman and Hall (2010)
15. Bouckaert, R.R.: Efficient AUC learning curve calculation. In: Australian Conference on Artificial Intelligence. Volume 4304 of Lecture Notes in Computer Science., Springer (2006) 181–191
16. Bayer, R.: Symmetric binary b-trees: Data structure and maintenance algorithms. Acta Inf. **1** (1972) 290–306
17. Brzezinski, D., Stefanowski, J.: Combining block-based and online methods in learning ensembles from concept drifting data streams. Information Sciences **265** (2014) 50–67
18. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis. J. Mach. Learn. Res. **11** (2010) 1601–1604

Learning from Imbalanced Data Using Ensemble Methods and Cluster-based Undersampling

Parinaz Sobhani[1, *], Herna Viktor[1], Stan Matwin[2]

[1] School of Electrical Engineering and Computer Science, University of Ottawa
{psobh090, hviktor}@uottawa.ca
[2] Faculty of Computer Science, Dalhousie University
stan@cs.dal.ca

Abstract. Imbalanced data, where the number of instances of one class is much higher than the others, are frequent in many domains such as fraud detection, telecommunications management, oil spill detection and text classification. Traditional classifiers do not perform well when considering data that are susceptible to both within-class and between-class imbalances. In this paper, we propose the ClustFirstClass algorithm that employs cluster analysis to aid classifiers when aiming to build accurate models against such imbalanced data sets. In order to work with balanced classes, all minority instances are used together with the same number of majority instances. To further reduce the impact of within-class imbalance, majority instances are clustered into different groups and at least one instance is selected from each cluster. Experimental results demonstrate that our proposed ClustFirstClass algorithm yields promising results compared to the state-of-the art classification approaches, when evaluated against a number of highly imbalanced datasets.

Keywords: Imbalanced data, Undersampling, Ensemble Learning, Cluster analysis

1 Introduction

Learning from data in order to predict class labels has been widely studied in machine learning and data mining domains. Traditional classification algorithms assume balanced class distributions. However, in many applications the number of instances of one class is significantly less than in the other classes. For example, in credit card fraud detection, direct marketing, detecting oil spills from satellite images and network intrusion detection the target class has fewer representatives compared to other classes. Due to the increase of these applications in recent years, learning in the presence of imbalanced data has become an important research topic.

It has been shown that when classes are well separated, regardless of the imbalanced ratio, instances can be correctly classified using standard learning algorithms [1]. However, having class imbalance in complex datasets results in the misclassification of data, especially of the minority class instances. Such data complexity covers issues such as overlapping classes, within-class imbalance, outliers and noise.

Within-class imbalance occurs when a class is scattered into smaller sub-parts representing separate subconcepts [2]. Subconcepts with limited representatives are called "small disjuncts" [2]. Classification algorithms are often not able to learn small disjuncts. This problem is more severe in the case of undersampling techniques. This is due to the fact that the probability of randomly selecting an instance from small disjuncts within the majority class is very low. These regions may thus remain unlearned. The main contribution of this paper is to address this issue by employing clustering techniques.

In this paper, a novel binary-class classification algorithm is suggested to handle data imbalance, mainly within-class and between-class imbalance. Our ClustFirstClass technique employs clustering techniques and ensemble learning methods to address these issues. In order to obtain balanced classes, all minority instances are used together with the same number of majority instances, as obtained after applying a clustering algorithm. That is, to reduce the impact of within-class imbalance majority instances are clustered into different groups and at least one instance is selected from each cluster. In our ClustFirstClass method, several classifiers are trained with the above procedure and combined to produce the final prediction results. By deploying several classifiers rather than a single classifier, information loss due to neglecting part of majority instances is reduced.

The rest of this paper is organized as follows. The next section presents related works for classification of imbalanced data. We detail our ClustFirstClass method in Section 3. Section 4 describes the setup and results of implementing and comparing of our algorithm with other state-of-the-art methods. Finally, Section 5 concludes the paper.

2 Related Work

Imbalanced class distribution may be handled by two main approaches. Firstly, there are sampling techniques that attempt to handle imbalance at data level by resampling original data to provide balanced classes. The second category of algorithms modifies existing classification methods at algorithmic level to be appropriate for imbalanced setting [3]. Most of the previous works in the literature have been concentrated on finding a solution at the data level.

Sampling techniques can improve classification performance in most imbalanced applications [4]. These approaches are broadly categorized as undersampling and oversampling techniques. The main idea behind undersampling techniques is to reduce the number of majority class instances. Oversampling methods, on the other hand, attempt to increase the number of minority examples to have balanced datasets. Both simple under- and oversampling approaches suffer from their own drawbacks. The main drawback of undersampling techniques is information loss due to neglecting part of majority instances. A major drawback of oversampling methods is the risk of overfitting, as a consequence of repeating minority examples many times.

In recent years, using ensemble approaches for imbalanced data classification has drawn lots of interest in the literature. Since ensemble algorithms are naturally designed to improve accuracy, applying them solely on imbalanced data does not solve the problem. However, their combination with other techniques such as under- and oversampling methods has shown promising results [16]. In [13] by integrating bagging with undersampling techniques better results are obtained. In [5], an ensemble algorithm, namely EasyEnsemble, has been introduced to reduce information loss. EasyEnsemble obtains different subsets by independently sampling from majority instances and combines each subset with all the minority instances to train base classifiers of the ensemble learner. In another work that extends bagging ensembles [20], the authors propose the use of so-called roughly balanced (RB) bagging ensembles, where the number of instances from the classes is averaged over all the subsets. A drawback of these bagging approaches is that they choose instances randomly, i.e. without considering the distribution of the data within each class while in [12] it has shown that one of the key factor in the success of ensemble method is majority instance selection strategy.

Cluster-based sampling techniques have been used to improve the classification of imbalanced data. Specifically, they have introduced “an added element of flexibility” that has not been offered by most of previous algorithms [4]. Jo et al. have suggested a cluster-based oversampling method to address both within-class and between-class imbalance [2]. In this algorithm, the K-means clustering algorithm is independently applied on minority and majority instances. Subsequently, each cluster is oversampled such that all clusters of the same class have an equal number of instances and all classes have the same size. The drawback of this algorithm, like most of oversampling algorithms, is the potential of overfitting the training data. In this paper, we also attempt to handle within and between class imbalances by employing clustering techniques. However, in our work we use undersampling techniques instead of oversampling, in order to avoid this drawback. In [6], a set of undersampling methods based on clustering (SBC) is suggested. In their approach, all the training data are clustered in different groups and based on the ratio of majority to minority samples in each cluster, a number of majority instances are selected from each cluster. Finally, all minority instances are combined with selected majority examples to train a classifier. Our approach is completely different as we only cluster majority instances and the same number of majority instances is selected from all clusters.

3 Proposed Algorithm

In this section, a new cluster-based under-sampling approach, called ClustFirstClass, is presented for binary classification. However, it can be easily extended to multiclass scenarios. This method is capable of handling between-class imbalance by having the same number of instances from minority and majority classes and within-class imbalance by focusing on all clusters within a class equally.

To have more intuition why clustering is effective for classification of imbalanced data, consider the given distribution of Figure 1. In this figure, circles represent majority class instances and squares are instances of minority class. Each of these classes contains several subconcepts. In order to have balanced classes, it follows that eight majority instances should be selected and combined with minority representatives to train a classifier. If these instances are randomly chosen, the probability of selecting an instance from region 1 and 2 will be low. Thus, the classifier will have difficulty classifying instances in these regions correctly. In general, the drawback of randomly selecting small number of majority class instances is that small disjuncts with less representative data may remain unlearned. By clustering majority instances in different groups and then selecting at least one instance from each cluster, this problem can be resolved.

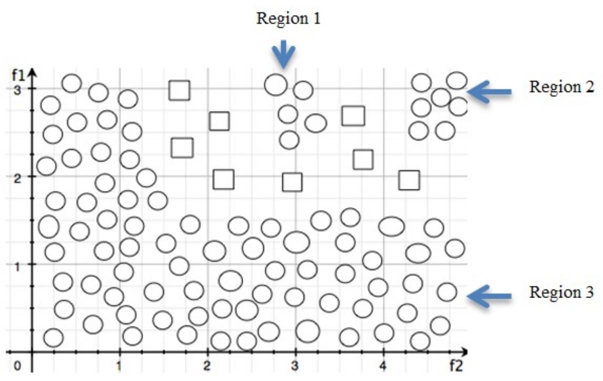


Fig. 1. A dataset with between and within class imbalance

3.1 Under-sampling based on clustering and K-nearest neighbor

In this group of methods, a single classifier is trained using all minority instances and equal number of majority instances. In order to have a representative from all subconcepts of the majority class, these instances are clustered into disjoint groups and one instance is selected from each cluster. However, rather than blindly selecting an instance, we attempt to choose more informative representative from each cluster. Principally, the difference between methods of this group is how these samples are selected from each cluster.

One of the most common representatives of a cluster is the cluster centroid. In our first suggested algorithm, clusters' centroids are combined with minority instances to train a classifier. For the rest of our methods, we follow the same procedures as presented in [7] to choose one instance from each cluster based on K-nearest neighbor (KNN) classifier. These three methods are widely used and have shown to produce good results in many domains [7]. Firstly, NearMiss1 selects the majority example from each cluster that has the minimum average distance to the three closest minority instances, as compared to the other examples in its cluster. In the same way, in NearMiss2, the example with minimum distance to its three farthest minority instances is chosen. The third alternative involves choosing the instance from each cluster that has the "most distance" to its three minority nearest neighbors.

3.2 Under-sampling based on clustering and ensemble learning

The main drawback of most undersampling methods, including those methods suggested earlier in this paper, is the information loss caused by considering a small set of majority instances and subsequently neglecting other majority class instances that may contain useful information for classification. Ensemble methods can solve this problem by using more instances of the majority class in different base learners [4]. In our proposed ensemble method, several classifiers are trained and combined to generate the final results. Each classifier is trained by selecting at least one sample from each cluster. Recall that the advantage of using cluster-based sampling instead of blind sampling is that all subconcepts are represented in training data. Therefore, none of them remains "unlearned".

The proposed ensemble algorithm is developed by training several base classifiers that are combined using a weighted majority voting combination rule, where the weight of each classifier is proportional to inverse of its error on the whole training set. Each learner is trained using *Dmin*, whole minority instances, and *Emaj*, selected majority instances, where *Emaj* contains |*Dmin*|/*k* randomly selected instances from each cluster. By assigning a value between 1 and |*Dmin*| to *k*, a balanced learner is obtained, while ensuring that instances from all subconcepts of majority class participate in training a classifier. The following pseudo-code describes our proposed algorithm in more details.

```
ClusFirstClass Algorithm
Input: $D =\{(x_i, y_i)\}$, i=1,…, N
Divide D into $D_{min}$ and $D_{maj}$
Cluster $D_{maj}$ into $k$ partition $P_i$ i=1,…,k
For each classifier $C_j$ j=1,…,m
    For each cluster $P_i$
        $E_{maj}$+= Randomly selected $|Dmin|/k$ instances of $P_i$
    End For
    Tr = $E_{maj} + D_{min}$
    Train $C_j$ using Tr
    $e_j$= Error rate of  $C_j$ on D
    $W_j$= log (1/ $e_j$)
 End For
Output: $C_{final}(x) = \text{argmax}_c \sum_{i=1}^{m} W_i\ |C_i(x) == c|$
```

4 Experiments and Results

In this section, first, common evaluation metrics for imbalanced data are introduced and then datasets and experimental setting that are used in this paper are presented.

Finally, our proposed algorithms are evaluated and compared with several state-of-the-art methods.

4.1 Evaluation Criteria

For imbalanced datasets, it is not sufficient to evaluate the performance of the classifier by only considering the overall accuracy [4]. In this paper, following other researchers, we use the F-measure and G-mean measures to evaluate the performance of different algorithms. Both F-measure and G-mean are functions of confusion matrix, a popular representation of the classifier performance. The F-measure considers both precision and recall at the same time while G-mean combines sensitivity and specificity as an evaluation metric.

4.2 Datasets and Experimental Settings

In this section, first artificial and real datasets for our experiments are introduced and subsequently more details about our experimental settings are described. Our proposed algorithm is particularly effective in presence of within and between class imbalances. To evaluate efficiency of our proposed method, it is applied on two sets of artificial datasets with varying degree of between class imbalances and different number of subconcepts. Furthermore, it is tested on real datasets from UCI repository [9].

Table 1. Description of uni-dimensional artificial datasets

Imbalance ratio	**Dataset Size**	**0-0.25** +		**0.25-50** -		**0.50-0.75** +		**0.75-1** -	
1:9	80	4		68		4		4	
	400	20		340		20		20	
	1600	80		1280		80		80	
1:3	80	10		50		10		10	
	400	50		250		50		50	
	1600	200		1000		200		200	
Imbalance ratio	**Dataset Size**	**0-0.125** +	**0.125-0.25** -	**0.25-0.375** +	**0.375-0.50** -	**0.50-0.675** +	**0.675-0.75** -	**0.75-0.875** +	**0.875-1** -
1:9	80	2	23	2	23	2	3	2	23
	400	10	13	10	119	10	119	10	119
	1600	40	466	40	466	40	466	40	42
1:3	80	5	18	5	6	5	18	5	18
	400	25	27	25	91	25	91	25	91
	1600	100	366	100	366	100	366	100	102

To create artificial datasets with varying degree of imbalance ratio and the number of subconcepts, we follow the same procedure as [1] with one difference that majority class as well as minority class has small disjuncts. In our artificial datasets, majority class instances have at least one small disjuncts. As in [1], three parameters are considered to create different datasets: dataset size, number of subconcepts and the imbalance ratio. Two sets of artificial datasets one uni-dimensional and the other multi-dimensional are generated.

Table 1 describes the number and label of data in each subconcept in uni-dimensional space. Data are distributed uniformly in intervals. For datasets with eight subconcepts, one of the intervals of majority data (negative label) is selected randomly as small disjunct with less representative data. Multi-dimensional datasets have five dimensions and we have the same clusters as [1]. The definition of subconcepts and dataset sizes is the same as described datasets in table 1.

In [8], a benchmark of highly imbalanced datasets from UCI repository is collected and prepared for binary classification task. We selected 8 datasets with wide range of imbalance ratios (from 9 to 130), sizes (from 300 to over 4000 examples) and attributes (purely nominal, purely continuous and mixed) from this benchmark. Table 2 shows the summary of these datasets. Here, all the nominal features have been converted to binary values with multiple dimensions. Following [8], datasets that had more than two classes have been modified by selecting one target class as positive, and considering the rest of the classes as being negative. Continuous features have been normalized to avoid the effect of different scales for different attributes especially for our distance measurements.

Table 2. The Summary of Datasets. In Features, N and C represent Nominal and Continuous respectively.

Dataset	Size	Features	Target	Imbalance ratio
Ecoli	336	7C	imU	1:9
Spectrometer	531	93C	LRS >=44	1:11
Balance	625	4N	Balance	1:12
Libras Move	360	90C	Positive	1:14
Arrhythmia	452	73N, 206C	Class=06	1:17
Car Eval.	1728	6N	Very good	1:25
Yeast	1484	8C	ME2	1:28
Abalone	4177	1N, 7C	Ring=19	1:130

All algorithms are implemented in the MATLAB framework. In all experiments, 5-fold stratified cross validation is applied. 5-fold cross validation is chosen due to limited number of minority instances in most datasets. The whole process of cross validation is repeated ten times and the final outputs are the means of these ten runs.

Decision trees have been commonly used in several imbalanced problems as a base classifier [5], [11], [12]. In this paper, the CART algorithm [10] is chosen as base learning method for our experiments.

We applied the K-means clustering algorithm to partition majority instances. However, instead of using the Euclidean distance to find similarity of instances, the L1-norm is used. The advantage of using the L1-norm over the Euclidean distance is that it is less sensitive to outliers in the data. Also, the probability of having a singleton partition for outliers is less than Euclidean distance [14]. Further, it has been shown that using the L1- norm is more suitable when learning in a setting which is susceptible to class imbalance, especially where the number of features is higher that the number of minority class examples [18].

4.3 Results and Analyses

In the first experiment, we evaluate the performance of our proposed single classifiers. Table 3 demonstrates the results of applying these methods on our real datasets and comparing them in terms of F-measure and G-mean. The results show that NearMiss1 has better performance compared to other classifiers and the classifier that uses the centroids as cluster representative has significantly lower F-measure and G-means. It can be concluded that cluster centroids are not informative for our classification task. In summary, as we expected, single undersampling learner suffers from information loss. In the rest of the experiments, we use an ensemble-learning method instead of using a single CART classifier.

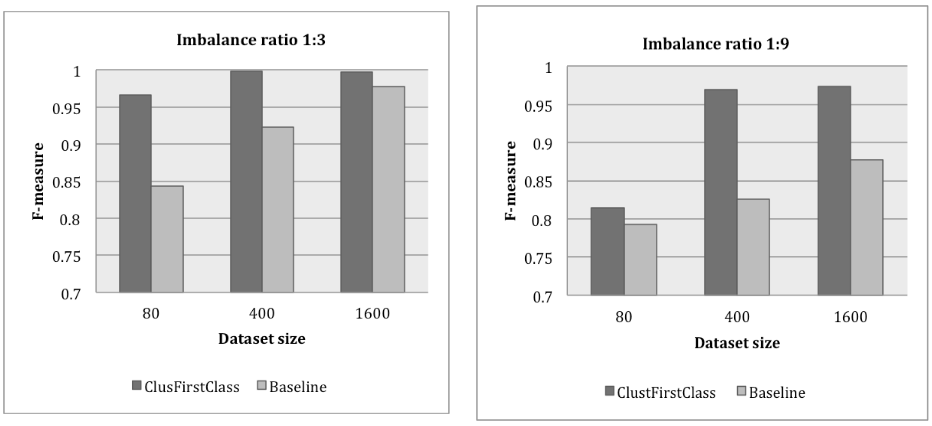
In the next experiment, to evaluate our proposed ensemble classifier ClusFirstClass in different scenarios with different degree of within and between class imbalances, it is applied on artificial datasets. We consider a bagging ensemble learner that chooses randomly a subset of majority instances to be combined with all minority instances, as the baseline and compare the performance of this algorithm with our proposed method. The only difference between baseline method and ClusFirstClass is that it chooses majority instances randomly. It has the same number of base learners and combination rule.

Table 3. F-measure and G-mean of proposed single classifiers

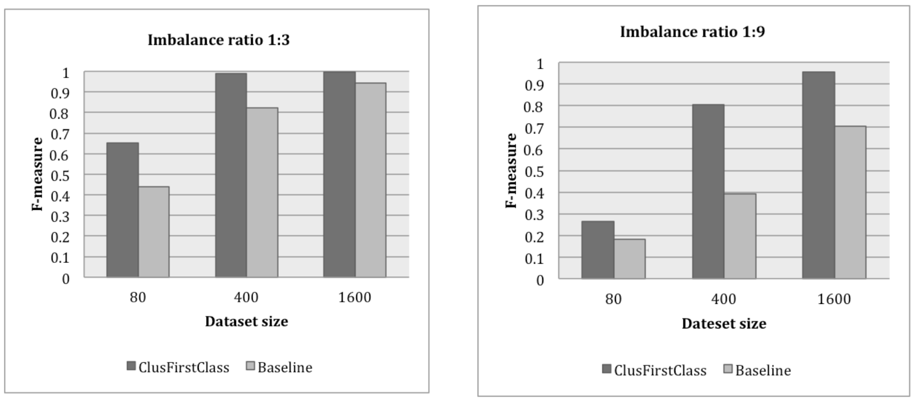
Dataset	**F-Measure**				**G-Mean**			
	Near Miss1	Near Miss2	Most-Distant	Centroid	Near Miss1	Near Miss2	Most-Distant	Centroid
Ecoli	0.5915	0.5671	0.5392	0.5740	0.8433	0.8261	0.8357	0.8527
Spectrometer	0.4874	0.5180	0.4746	0.4709	0.8488	0.8445	0.8488	0.8488
Balance	0.1448	0.1417	0.1802	0.1415	0.5050	0.4600	0.5551	0.0581
Libras Move	0.3945	0.3433	0.3154	0.3789	0.8147	0.7865	0.7708	0.7991
Arrhythmia	0.3594	0.3074	0.3342	0.3646	0.7855	0.7411	0.7623	0.7737
Car Eval.	0.8313	0.8057	0.2778	0.2394	0.9730	0.9857	0.8907	0.8513
Yeast	0.2398	0.1999	0.1928	0.1138	0.7827	0.7657	0.7872	0.5412
Abalone	0.0301	0.0328	0.0275	0.0174	0.6496	0.6820	0.6599	0.2873
Average	**0.3849**	0.3645	0.2927	0.2876	**0.7753**	0.7614	0.7638	0.6265

Figure 2 and 3 illustrates the results of applying our proposed method on previously described uni-dimensional and multi-dimensional artificial datasets respectively. In all 12 scenarios ClusFirstClass is compared to baseline method in terms of F-measure. For all datasets, our proposed method has considerably better performance compared to baseline. In most cases, as the imbalance ratio and the number of subconcepts increase the difference between our proposed classifier and baseline algorithm becomes more significant.

In the first experiment with proposed single classifiers, we have clustered majority instances into k groups using the k-means algorithm, where *k*=|*Dmin*|. In experiments on artificial datasets, obviously the number of clusters is equal to the number of subconcepts within the majority class. For the next experiments, in order to compute the natural number of clusters, different number of k from 1 to |*Dmin*| is tested to find the one with the best average Silhouette plot [17].



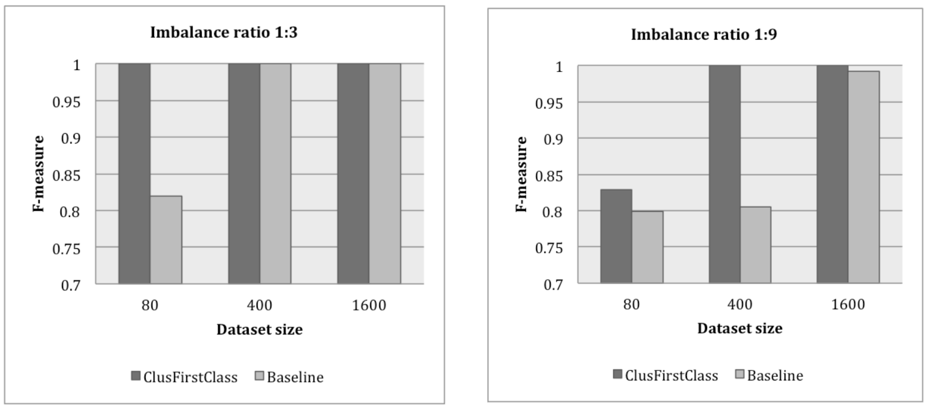
a) In the case of having four subconcepts



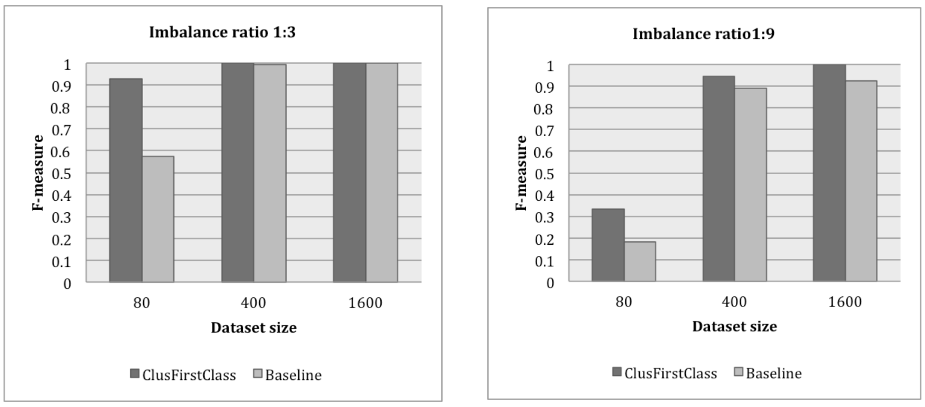
b) In the case of having eight subconcepts

Fig. 2. Results of applying our proposed method on previously described uni-dimensional artificial datasets in terms of F-measure

Table 4 shows the results of comparing our proposed undersampling ensemble algorithm based on clustering with another undersampling ensemble method, EasyEnsemble [5] and two cluster-based algorithms, Cluster-based oversampling [2] and SBC [6]. Our algorithm outperforms other undersampling ensemble methods on almost all datasets in terms of F-Measure and G-Mean. Compared to the cluster-based oversampling, although that method achieves better F-measure on two (out of 9) datasets, the averaged F-measure and G-Mean of our algorithm is better than that of Cluster-based oversampling. Clust-First-Class outperforms SBC on all datasets.



a) In the case of having four subconcepts



b) In the case of having eight subconcepts

Fig. 3. Results of applying our proposed method on previously described multi-dimensional artificial datasets in terms of F-measure

5 Conclusion and Future Work

In this paper, we introduce a new cluster-based classification framework for learning from imbalanced data. In our proposed framework, first majority instances are clustered into k groups and then at least one instance from each cluster is selected to combine with all minority instances, prior to training. This approach is capable of handling between-class imbalance by selecting approximately the same number of instances from minority and majority classes. Further, we address within-class imbalance by focusing on all clusters equally. Finally, to reduce information loss due to choosing small number of majority instances in highly imbalanced datasets, we employ an ensemble learning approach to train several base learners with different subsets of majority instances. An advantage of our ClustFirstClass method is that we guide the selection of majority instances used during training, as based on the clusters obtained by the k-means algorithm

To evaluate the efficiency of our proposed method, it is applied on two sets of artificial datasets with varying degree of between class imbalances and different number of subconcepts. For all datasets, our proposed method has considerably better performance compared to the baseline method. In most cases, as the imbalance ratio and the number of subconcepts increase, the difference between our proposed classifier and baseline algorithm becomes more significant. Experimental results on real datasets demonstrate that our proposed ensemble learner has better performance than our proposed single classifiers. It also shows that our suggested ensemble method yields promising results compared to other state-of-the-art methods in terms of G-means and F-measure.

Several directions of future research are open. Our experimental results indicate that using the K-means algorithm yield encouraging results. However, we are interested in exploring other cluster analysis algorithms, since the K-means algorithm may not be ideal when considering highly imbalanced datasets [19], or when considering extending our work to the multi-class problems. Thus, we plan to investigate the use of more sophisticated clustering algorithms to partition the majority instances. Another direction would be to consider other ensemble-based techniques. In particular, ECOC [15] may be a favorable choice as it targets performance improvement in a binary classification setting. We also plan to extend our experiments with more datasets and compare it with more ensemble algorithms such as RB bagging [20] and also testing other base learning algorithms such as SVM and KNN.

Table 4. F-measure and G-mean of proposed ensemble classifier, ClustFirstClass, compared to EasyEnsemble, Cluster-oversampling and SBC methods

Dataset	**F-Measure**				**G-Mean**			
	Clust First Class	Easy Ensemble	Clust Over Sample	SBC	Clust First Class	Easy Ensemble	Clust Over Sample	SBC
Ecoli	**0.5961**	0.5612	0.5088	0.5140	**0.8689**	0.8658	0.6899	0.8489
Spectrometer	0.5944	**0.6924**	0.6740	0.4485	0.8878	**0.9064**	0.8053	0.8186
Balance	**0.1524**	0.0793	0.0290	0.1508	**0.5223**	0.3452	0.0967	0.4971
Libras Move	0.5912	0.4806	**0.6652**	0.4258	**0.8451**	0.8407	0.8006	0.7372
Arrhythmia	**0.7475**	0.6360	0.5757	0.5996	**0.9489**	0.8802	0.7548	0.9219
Car Eval.	**0.8331**	0.3613	0.9566	0.6892	**0.9918**	0.9237	0.9812	0.9792
Yeast	0.2720	0.2613	**0.2798**	0.2065	**0.8054**	0.8044	0.5095	0.8016
Abalone	**0.0449**	0.0381	0.0618	0.0315	**0.7446**	0.7309	0.1903	0.6794
Average	**0.4790**	0.3888	0.4689	0.3832	**0.8267**	0.7872	0.6035	0.7855

References

1. Japkowicz N., Stephen S.: The class imbalance problem: A systematic study. J. Intelligent Data Analysis 6(5), 429–450 (2002)
2. Jo T., Japkowicz N.: The class imbalance versus small disjuncts. ACM SIGKDD Exploration Newsletter 6(1), 40-49 (2004)
3. Sun Y., Kamel M.S., Wong A.K.C., Wang Y.: Cost-Sensitive Boosting for Classification of Imbalanced Data. J. Pattern Recognition 40(12), 3358-3378 (2007)
4. He H., Garcia E.: Learning from imbalanced data. J. IEEE Transactions on Data and Knowledge Engineering 9(21), 1263–1284 (2009)
5. Liu X.Y., Wu J., Zhou Z.H.: Exploratory Under Sampling for Class Imbalance Learning. Proc. Int'l Conf. Data Mining, pp. 965-969, (2006)
6. Yen, Lee: Cluster-based under-sampling approaches for imbalanced data distributions, Expert Systems with Applications: An International Journal Volume 36 Issue 3, April, (2009)
7. Zhang J., Mani I.: KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction. Proc. Int'l Conf. Machine Learning (ICML '2003), Workshop Learning from Imbalanced Data Sets (2003)
8. Ding, Zejin: Diversified ensemble classifiers for highly imbalanced data learning and its application in bioinformatics. Phd thesis, Georgia State University (2011)
9. Bache K., Lichman M.: UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science (2013)
10. Breiman L., Friedman J., Olshen R., Stone C.: Classification and Regression Trees. Boca Raton, FL: CRC Press (1984)
11. Batista G., Prati RC, Monard MC.: A study of the behaviour of several methods for balancing machine learning training data. SIGKDD Explor 6(1), 20–29 (2004)
12. Bianchi N. et al.: Synergy of multi-label hierarchical ensembles, data fusion, and cost-sensitive methods for gene functional inference, Machine Learning 88(1), 209-241 (2012)
13. Błaszczyński J. et al.: Extending Bagging for Imbalanced Data. In *Proceedings of the 8th International Conference on Computer Recognition Systems,* pp. 269-278 (2013)
14. Manning C., Schutze H.: Foundations of Statistical Natural Language Processing. MIT Press Cambridge, MA (1999)
15. Dietterich T. G., Bakiri G.: Solving Multiclass Learning Problems via Error-Correcting Output Codes. J. AI Research 2, 263-286 (1995)
16. Galar M. et al: A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches Systems. Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions, 42(4) (2012)
17. Rousseeuw Peter J.:Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. Computational and Applied Mathematics 20, 53–65 (1987)
18. Ng A.: Feature selection, L1 vs. L2 regularization and rotational invariance. 21st International Conference on Machine Learning (2004)
19. Coates A., Ng A.: Learning Feature Representations with K-means, In Neural Networks: Tricks of the Trade (Second Edition), Springer LNCS 7700, 561-580 (2012)
20. Shohei H., Hisashi K., Yutaka T.: Roughly balanced bagging for imbalanced data, Statistical Analysis and Data Mining, 2, 5-6, 412-419 (2009)

Learning Complex Activity Preconditions in Process Mining

S. Ferilli[1,2], B. De Carolis[1], G. Fatiguso[1], and F. Esposito[1,2]

[1] Dipartimento di Informatica – Università di Bari
firstname.lastname@uniba.it
[2] Centro Interdipartimentale per la Logica e sue Applicazioni – Università di Bari

Abstract. The availability of automatic support may determine the successful accomplishment of many real-world procedures. However, the underlying process models are too complex for writing and setting up them manually, and even standard machine learning approaches may be unable to infer them. Additionally, suitable conditions may that determine whether some tasks are to be carried out or not. These conditions may be in turn very complex, involving sequential relationships that take into account the past history of the process.
This paper presents a First-Order Logic approach to learn complex process models extended with conditions. It combines two powerful Inductive Logic Programming systems. The overall system was applied to learning the daily routines of the user of a smart environment, for predicting his needs and comparing the actual situation with the expected one. Promising results have been obtained, that proved its efficiency and effectiveness, and with a domain-specific dataset.

1 Introduction

Many real-world procedures are nowadays so complex that automatic supervision and support may be determinant for their successful accomplishment. This requires providing the automatic systems with suitable process models. However, these models are too complex for writing and setting up them manually, and even standard machine learning approaches may be unable to infer them. The expressive power of these models can be enhanced by allowing them to set suitable conditions that determine whether some tasks are to be carried out or not. These conditions may be in turn very complex, involving sequential relationships that take into account the past history of the process.

This paper presents an approach to learn complex process models extended with complex conditions on their components. It works in First-Order Logic (FOL), that allows to express in a single formalism both the models and the associated conditions. FOL allows automatic reasoning and learning with structured representations that are able to represent and handle relationships among the involved entities and their properties. These capabilities go beyond traditional representations, where any description must consist of a fixed number of values, such as feature vectors. Our approach is based on the combination of two powerful learning systems: WoMan [10, 6], that is in charge of learning the process model, and InTheLEx [5], that is in charge of learning the conditions. Compared to [6], here we present for the first time in detail the interaction between WoMan and InTheLEx, especially as regards the learning and exploitation of conditions involving (possibly multidimensional) sequential information.

Let us introduce some preliminary notions and terminology. A *process* is a sequence of *events* associated to actions performed by agents [3]. A *workflow* is a (formal) specification of how a set of tasks can be composed to result in valid processes, often modeled as directed graphs where nodes are associated to states or tasks, and edges represent the potential flow of control among activities. It may involve sequential, parallel, conditional, or iterative executions [18]. Each task may have pre- and post-conditions, which determine whether they will be executed or not [1]. An *activity* is the actual execution of a task. A *case* is a particular execution of actions in a specific order compliant to a given workflow [12]. *Process Mining* [19]) aims at inferring workflow models from examples of cases. *Inductive Logic Programming* (ILP) [17] is the branch of Machine Learning based on FOL as a representation language.

The rest of this paper is organized as follows. Section 2 introduces the representation formalism on which our proposal is based. Then, Section 3 shows how process models are learned, and 4 describes how they are exploited. Section 5 presents experiments that show the effectiveness of the proposed approach. Lastly, Section 6 concludes the paper and outlines future work directions.

2 Representation

When tracing process execution, events are usually listed as sequences of 6-tuples (T, E, W, P, A, O) where T is a timestamp, E is the type of the event (begin process, end process, begin activity, end activity), W is the name of the workflow the process refers to, P is a unique identifier for each process execution, A is the name of the activity, and O is the progressive number of occurrence of that activity in that process [1, 12]. If contextual information is to be considered, for inferring conditions on the model, we may assume that an additional type of event 'context' can be specified, in which case A contains the logic atoms that describe the relevant context. The predicates on which such atoms are built are domain-dependent, and are defined by the knowledge engineer that is in charge of setting up the reasoning or learning task.

E.g., an excerpt of a 'sunday' daily-routine workflow case trace might be:
(201109280900, begin_process, sunday, c3, start, 1)
(201109280900, context, sunday, c3, [john(j),happy(j),hot_temp], 1)
(201109280900, begin_activity, sunday, c3, wake_up, 1)
(201109280905, end_activity, sunday, c3, wake_up, 1)
(201109280908, begin_activity, sunday, c3, toilet, 1)
(201109280909, context, sunday, c3, [radio(r),status(r,rs),on(rs),listen(j,r)], 1)
(201109280909, begin_activity, sunday, c3, shower, 1)

...
(201109282221, end_process, sunday, c3, stop, 1)

The activity-related 6-tuples in a case trace can be translated into FOL as a conjunction of ground atoms built on the following reserved predicates [10]:

`activity(`S`,`T`)` : at step S task T is executed;
`next(`S'`,`S''`)` : step S'' follows step S'.

Steps represent relevant time points in a process execution; they are derived from event timestamps, and are denoted by unique identifiers. This formalism allows to explicitly represent parallel executions in the task flow. So, here sequentiality is not restricted to be a linear relationship: rather than being simple 'strings' of atoms, our representations induce a Directed Acyclic Graph. Given a FOL case description D and one of its steps $\overline{s}$ (i.e., $\exists t$ s.t. `activity(`$\overline{s}$`,`t`)` $\in D$), the presence in D of many `next(`$\overline{s}$`,`s_i`)` atoms indicates that the execution of $\overline{s}$ is followed by the parallel execution of many other tasks corresponding to steps s_i. The presence in D of many `next(`s_j`,`$\overline{s}$`)` atoms indicates that the parallel execution of the tasks corresponding to the s_j's converges to the execution of $\overline{s}$. A sequential execution, having just one input task and one output task, is the special case of a single s_i (or s_j).

In addition to the flow of activities, many other kinds of sequential information may be relevant in a process. So, we allow our descriptions to include sequential relationships along several dimensions (e.g., time, space, etc.). We call *events* the terms in a FOL description on which sequential relationships can be set (so, steps are a special kind of events). Just like any other object, events may have properties and relationships to other events and/or objects. We reserve the following predicate to express sequential information among events [9]:

`next(`I_1`,`I_2`,`D`)` : event I_2 immediately follows event I_1 along dimension D.

In this representation, the sequential relationship among steps becomes just one of the allowed dimensions. We consider it as the *default* dimension, so we still use for it the `next/2` predicate, without an explicit dimension argument.

For instance, let us assume that the execution of activities and the sensing of the context are asynchronous. Then, we may use an independent dimension *context* for the flow of contextual information, and associate each activity-related event to the corresponding context description using a `context/2` predicate.

Thus, the FOL translation of the previous sample trace might start as follows:

$$\texttt{activity}(s_b,\texttt{start}),\ \texttt{start(start)},\ \texttt{context}(s_b,c_0),\ \texttt{john}(j),\ \texttt{happy}(c_0,j),\\ \texttt{hot\_temp}(c_0),\ \texttt{next}(s_b,s_0),\ \texttt{activity}(s_0,\texttt{wake\_up}),\ \texttt{wake\_up(wake\_up)},\\ \texttt{context}(s_0,c_0),\ \texttt{next}(s_0,s_1),\ \texttt{activity}(s_1,\texttt{toilet}),\ \texttt{toilet(toilet)},\\ \texttt{context}(s_1,c_0),\ \texttt{next}(c_0,c_1,\mathit{context}),\ \texttt{radio}(c_1,r),\ \texttt{status}(c_1,r,rs),\\ \texttt{on}(c_1,rs),\ \texttt{listen}(c_1,j,r),\ \texttt{next}(s_0,s_2),\ \texttt{activity}(s_2,\texttt{shower}),\\ \texttt{shower(shower)},\ \texttt{context}(s_2,c_1),\ \ldots,\ \texttt{activity}(s_e,\texttt{stop})$$

to be read as: "The first activity in this Sunday process is *wake_up*, that takes place in the initial context, where the temperature is hot and John is happy. The next activity, *toilet*, follows *wake_up* and takes place in the same context. Then, the context changes, with the user listening to the radio, which is on. While *toilet* is still running, the new parallel activity *shower* starts, associated to the context in which the user is still listening to the radio." And so on...

The structure of a workflow is expressed as a set (to be interpreted as a conjunction) of atoms built on the following predicates:

`task(`t`,`C`)` : task t occurs in cases C, where C is a multiset of case identifiers (because a task may be carried out several times in the same case);

`transition(`I`,`O`,`p`,`C`)` : transition p, that occurs in cases C (again a multiset), consists in ending all tasks in I and starting all tasks in O.

In the previous example, we might have:

`task(start,{c3,...}).`	`transition({start},{wake_up},t0,{c3,...}).`
`task(wake_up,{c3,...}).`	`transition({wake_up},{toilet,shower},t1,{c3,...}).`
`task(toilet,{c3,...}).`	`...`
`task(shower,{c3,...}).`	
`...`	
`task(stop,{c3,...}).`	

As regards conditions, each `activity(`s`,`t`)` atom in the case description generates an observation, to be used as a training example during the learning phase, or as a test one during the monitoring phase. For the pre-conditions, the observation includes only the atoms in the description associated to steps up to s. In the 'sunday' case, `activity(`s_2`,shower)` yields:

shower(s_2) :- activity(s_b,start), start(start), context(s_b,c_0), john(j), happy(c_0,j), hot_temp(c_0), next(s_b,s_0), activity(s_0,wake_up), wake_up(wake_up), context(s_0,c_0), next(s_0,s_1), activity(s_1,toilet), toilet(toilet), context(s_1,c_0), next(c_0,c_1,*context*), radio(c_1,r), status(c_1,r,rs), on(c_1,rs), listen(c_1,j,r), next(s_0,s_2), activity(s_2,shower), shower(shower), context(s_2,c_1).

Sequential relationships are transitive. So, while observations are always expressed in terms of `next/3` and `next/2` relationships, conditions are expressed in terms of a more general sequential relationship `after/3`:

`after(`I_1`,`I_2`,`D`)` : I_2 (possibly indirectly) follows I_1 along dimension D.

representing the transitive closure of immediate adjacency, and defined as:

after(X,Y,D) :- next(X,Y,D).
after(X,Y,D) :- next(X,Z,D), after(Z,Y,D).

Thus, a precondition learned from the previous example might be:

shower(X) :- activity(X,Y), shower(Y), context(X,Z), radio(Z,T), status(Z,T,W), on(Z,W), after(U,X,*default*), start(U), context(U,S), hot_temp(S), after(U,V,*default*), activity(V,R), toilet(R), context(V,S), after(S,Z,*context*).
(in order to have a shower, the radio must be on, and the *toilet* activity must have started when the temperature was hot).

Again, in this representation sequentiality is not restricted to be a linear relationship, and induces a Directed Acyclic Graph for each dimension.

3 Learning

Given the FOL description D of a case c, a model is built or refined as follows:

1. For each `activity(`s`,`t`)` atom in D,
   (a) if an atom `task(`t`,`C`)` exists in the current model, then replace it by `task(`t`,`$C \cup \{c\}$`)`; otherwise, add a new atom `task(`t`,`$\{c\}$`)` to the current model.
2. For each `next(`s'`,`s''`)` atom in D, indicating the occurrence of a transition,
   (a) collect from c the multisets I and O of the input and output task(s) of that transition (respectively)[1],
   (b) if an atom `transition(`I`,`O`,`p`,`C`)` having the same inputs and outputs exists in the current model, then replace it by `transition(`I`,`O`,`p`,`$C \cup \{c\}$`)`; otherwise, create a new atom `transition(`I`,`O`,`p`,`$\{c\}$`)`, where p is a fresh transition identifier.
   (c) remove from D the `next/2` atoms used for the transition (to avoid that the same occurrence of the transition is detected for each of such atom).

Differently from all previous approaches, this technique is *fully incremental.* It can start with an empty model and learn from one case, while others need a large set of cases to draw significant statistics. It can refine an existing model according to new cases whenever they become available, introducing alternative routes (alternative executions, represented by different `transition/4` atoms having the same I argument, may emerge from the analysis of several cases) or even adding new tasks if they were never seen in previous cases. This ensures continuous adaptation of the learned model to the actual practice, carried out efficiently, effectively and transparently to the users. Noisy data can be handled naturally by the learned models. Indeed, the probability of a transition is proportional to the number of cases in which it occurred in the training cases. Each transition stores the multiset of cases in which it occurred, and the multiset is updated each time a case is processed. Thus, the weight of a transition is simply the ratio of the cardinality of its associated multiset over the number of training cases.

While learning the workflow structure for a given case, examples for learning task pre-conditions are generated as well, and provided to the ILP system. InTheLEx [5] was chosen both for its compliance with the fully incremental approach to learning the workflow structure, and because it is also endowed with a positive-only-learning feature (the typical setting in Process Mining). It was extended to handle sequential information according to the technique presented in [9], which is peculiar in the current literature. While most works have focused on sequential information on a single dimension [13, 14, 15, 16, 2, 11], it works in a multidimensional setting, and allow complex interrelationships among any mix of events and involved objects. This means, for instance, that events in different dimensions can be related to each other, which prevents simple extension

[1] Cases in which $|I| > 1$ and $|O| > 1$ represent complex situations where multiple activities are needed to fire several new activities in the next step, which are not handled by other systems in the literature.

of single-dimension approaches to multiple orthogonal dimensions. Also, it goes beyond strictly linear sequences requiring a total ordering relationship among events, and allows for 'parallel' events along the same dimension. Moreover, it allows to learn rules in the classical ILP fashion, but allowing preconditions to include sequential information, while other works classify sequences [13], or infer predictive models for them [14, 15, 2], or extract frequent patterns [16, 4].

Given two examples for learning conditions involving sequential information described according to the formalism above, their generalization (in a positive-examples-only setting no specialization is ever needed) is determined as follows:

1. set an association between a subset of events in the former description and corresponding events in the latter;
2. generalize the corresponding sequential relationships;
3. generalize the rest of the descriptions consistently with the result of (2).

Since one is usually interested in generalizations fulfilling particular properties (e.g., the least general ones), an optimization problem is cast where all possible such generalizations must be computed for identifying the best one.

Unfortunately, step 1 alone introduces a significant amount of indeterminacy (i.e., many portions of one description map onto many portions of the other): given two sequences of events $S' = \langle s'_i \rangle_{i=1,\dots,n}$ and $S'' = \langle s''_j \rangle_{j=1,\dots,m}$, with $n \leq m$, there are $\sum_{k=1}^{n} \binom{n}{k} \cdot \binom{m}{k}$ possible associations to be checked. It is clearly unpractical. Thus, a heuristic is used, that directly selects a single, most promising association to be exploited as a base for steps 2 and 3. It is based on the intuition that two events should be associated if they are similar to each other, and that the best association should obtain the highest overall similarity among all the possible associations. First of all, a description for each event is obtained as the set of literals that are in a neighborhood of at most i hops from literals involving that event (where a hop exists between literals that share at least one argument). Then, the overall association is obtained using a greedy approach: the similarity of all pairs of descriptions of events, one from each clause, is computed using the measure proposed in [7]; the pairs are ranked by decreasing similarity, and the rank is scanned top-down, starting from the empty generalization and progressively extending it by adding the generalization of the descriptions of each pair whose association (involving both events and other objects) is compatible with the cumulative association of the generalization computed so far.

Once the pairs of associated events in the two clauses have been determined, the corresponding sequential predicates are generalized in step 2. First, we simplify the sequential descriptions, removing all the sub-sequences of useless intermediate events that have not been associated and replacing them with 'compound' `after` atoms. Then, these simplified atoms can be generalized.

Finally, in step 3, two clauses are created, each having as a body all non-sequential literals not used in the event generalization step. These two clauses are generalized using the standard (non-sequential) algorithm proposed in [7]. Since all the information concerning events was fixed in the previous steps, this generalization is only in charge of finding the best mapping among the remaining literals. This introduces additional indeterminacy.

4 Exploitation

The learned model can be used to monitor a new case and check its compliance, as described in [6]. As long as the 6-tuples of the case trace are fed to the system, they are used to build the corresponding FOL case description, and compared to the model as follows, depending on the type of event.

begin_process : load the corresponding process model; set the current marking to {start}; pending transitions and pending/finished activities are empty.

begin_activity : add the activity to the set of pending activities; then, check that the pre-conditions for that activity are satisfied, in which case:

- if the new activity belongs to the pending tasks of a pending partial transition, delete the activity from the pending partial transition
- if the new activity is in the output tasks of a transition whose input tasks are all satisfied by the current marking, the transition is not noisy, and its pre-conditions are satisfied, then 'apply' the transition: delete its input tasks from the current marking and add the output tasks other than the activity (if any) to a new pending transition.

If the new activity neither belongs to a pending partial transition nor to the output tasks of an enabled transition, or if any of the pre-conditions is not satisfied, then it is not compliant with the model and a warning is raised. Note that, for some models, there might be many valid options (i.e., several partial transitions to be completed and/or transitions that are enabled by the current marking). In this case all possible alternatives must be carried to the next step, and then filtered out when they turn out to be incompatible with the rest of the execution. An upper bound to the number of repetitions of loops (if any) can be set (e.g., as the maximum number of repetitions encountered in the training cases).

end_activity : if the activity is in the set of pending activities then delete it and add a corresponding token to the current marking, otherwise raise an error.

end_process : if an error was raised by previous events, or the current marking is not empty, then raise an error; otherwise the FOL case description can be used to refine the model (if no warning was raised by previous events, then the refinement just affects the task statistics; otherwise, it causes a change in the model structure and/or preconditions).

Noisy transitions are those whose associated multiset of cases C_t, occurred in a number of cases that represents a fraction of all n training cases less than the allowed noise threshold $N \in [0, 1]$ (i.e., $|C_t|/n < N$). Indeed, N represents the minimum frequency threshold under which transitions are to be ignored.

Conditions are checked against the current context and status of the process. Our approach works by associating the contextual information first, and then completing the coverage with the sequential information, as follows:

1. apply preliminary coverage check to the non-sequential (i.e., contextual and cross-event) part of the description, obtaining a covering association E also including event bindings (there may be many);

2. complete the coverage check: For all atoms `after(`X`,`Y`,`D`)` in the model:
   (a) pick the events s and t in the observation associated to X and Y, respectively, using the event association fixed by E;
   (b) check that the observation includes a sequence from s to t along dimension D, not involving other events that have been fixed by E (if any, this must be unique); if this check fails, backtrack on (1) to find another covering association.

Step 1 can exploit existing (efficient) coverage procedures for non-sequential representations. As regards sequence checking, we need to find all paths between two events in the directed graph induced by the sequence, where nodes are events and edges connect events between which a sequence atom is present in the description. This operation can be optimized as follows. Let us consider the set I of sequence atoms in the observation concerning a fixed dimension.

1. create the sequentiality graph G induced by I;
2. compute the topological sort T of G (i.e., the list of nodes in I in which a node u appears after a node v if there exists a path from v to u in G);
3. Among all associations $A \subseteq E$ between events in the model and events in the observation, by which the model covers the observation (considering non-sequential atoms only), find at least one for which the sequential part is covered, to be checked as follows:
   (a) For all sequence atoms `after(`X`,`Y`,`D`)` in the model:
      i. pick events s and t associated to X and Y, respectively, by A;
      ii. extract from T the sublist $S = [s, ..., t]$ (if t does not follow s in T then S is empty, and hence this step fails);
      iii. driven by the sequence of events in S, check whether in I there exists a chain of sequence atoms that leads from s to t.

5 Evaluation

The proposed techniques were implemented in YAP Prolog 6.2.2, and tested on a notebook PC endowed with an Intel Dual Core processor (2.0 GHz), 4 GB RAM + 4 GB SWAP, and Linux Mint 13 operating system. We evaluated our approach in a Process Mining task aimed at learning user's daily routines in a Smart Environment domain [8]. The learned model will be used to predict his needs, so that the environment may provide suitable support, and to compare the actual situation with the expected one, in order to detect and manage anomalies. For this purpose, we used a real-world dataset taken from the CASAS repository (`http://ailab.wsu.edu/casas/datasets.html`), concerning daily activities of people living in cities all over the world. In particular, we selected the Aruba dataset, involving an elderly person visited from time to time by her children. The Aruba dataset reports data concerning 220 days, represented as a sequence of timestamped sensor data, some of which annotated with a label indicating the beginning or end of a meaningful activity[2].

[2] Actually, for one day the activity labels were missing, for which reason the corresponding case was removed from the dataset.

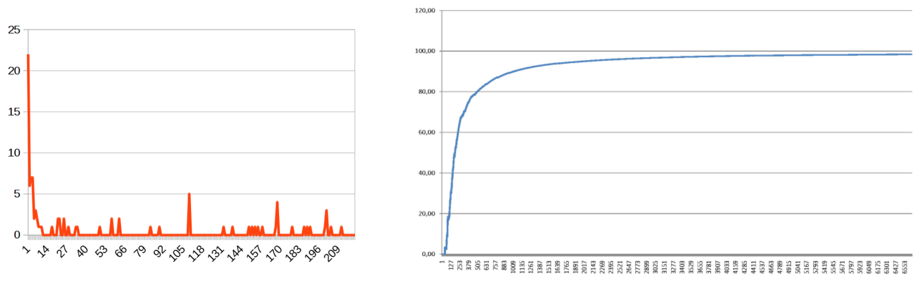


Fig. 1. Learning curves for the Aruba dataset: model (left) and pre-conditions (right)

We obtained a set of cases by splitting this dataset into daily cases according to the following logic: a new day starts after the last sleeping activity between midnight and noon. The 219 case descriptions were filled with contextual information coming from the status of the various sensors in the house. Three kinds of sensors are available: movement sensors (identified by prefix ‘m’), opening/closing sensors for doors and windows (identified by prefix ‘d’) and temperature sensors (identified by prefix ‘t’). Then, the resulting descriptions were used to learn both the process model and the preconditions for each task, i.e. regularities in context that were present in all executions of each task. The dataset involved 6530 activity instances (29.68 per day on average), each of which generated a pre-condition example. The average number of literals per pre-condition example was 739.85, with a minimum of 16 and a maximum of 2570.

We simulated a real setting in which the system starts from scratch, and learns the process model from the first day, progressively refining it as days go by. Each event is checked against the current model to assess its compliance. In case of non-compliance, a revision of the model is started. In the compliance check, since all examples are positive, there can be no False Positives nor True Negatives. Thus, as regards the predictive performance, Precision is always 1, and Accuracy is the same as Recall. The learning curves are reported in Figure 1. On the left, the curve shows how many non-compliant transitions were performed in each day. The fact that peaks become lower and sparser after day 7 confirms that the learned model converges to the ‘correct’ routine. The analogous curve for tasks is not shown, since it soon becomes flat at 0 after day 7. On the right, the curve shows the Accuracy of the pre-conditions learned. It passes 80% after about 600 examples (i.e., about 20 days), and goes on improving as days go by until 98.27%. The process model was learned in 497msec (2.27msec per day on average), including both the check and the learning effort if needed. This amounts to less than 0.1msec per activity on average. It involved 13 tasks and 96 transitions. The runtime for learning task pre-conditions was 263sec (0.04sec per example on average). This confirms that the approach can be applied on-line to the given environment, without causing unnecessary delays in the normal activities of the system or of the involved people.

Table 1. Preconditions generated by InTheLEx on the whole dataset

```
act_Sleeping(A) :- after(_,A,default), context(A,_), activity(A,_).
act_Meal_Preparation(A) :- after(_,A,default), context(A,_),
    activity(A,_).
act_Relax(A) :- context(A,_), activity(A,_).
act_Housekeeping(A) :- after(B,A,default), after(C,B,default),
    after(D,C,default), after(_,D,default), context(D,E),
    after(E,F,context), after(F,G,context), after(G,H,context),
    context(C,F), context(B,G), context(A,H), activity(A,_),
    activity(B,_).
act_Eating(A) :- after(_,A,default), context(A,B), status_m014(B,_),
    activity(A,_).
act_Wash_Dishes(A) :- after(_,A,default), context(A,B),
    after(C,B,context), after(_,C,context), activity(A,_).
act_Leave_Home(A) :- after(B,A,default), after(_,B,default), context(B,_),
    context(A,_), activity(A,_).
act_Enter_Home(A) :- after(B,A,default), act_Leave_Home(B),
    after(C,B,default), after(_,C,default), context(B,D),
    after(D,E,context), context(A,E), activity(B,_), activity(A,_).
act_Work(A) :- after(B,A,default), after(_,B,default), context(B,C),
    after(C,D,context), status_m026(D,E), on(E), context(A,D),
    activity(A,_).
act_Bed_to_Toilet(A) :- after(_,A,default), context(A,_), activity(A,_).
act_Resperate(A) :- after(B,A,default), after(_,B,default), context(B,C),
    after(C,D,context), status_m006(D,E), off(E), status_m008(D,F),
    off(F), status_m014(D,G), off(G), status_m018(D,H), off(H),
    status_m020(D,I), off(I), status_m021(D,J), off(J), status_m022(D,_),
    status_m026(D,_), status_m028(D,_), status_m013(C,_), status_m018(C,_),
    status_m019(C,_), status_m020(C,_), status_m021(C,_), context(A,D),
    activity(B,_), activity(A,_).
```

The task preconditions learned by InTheLEx are reported in Table 1. According to these rules, activity Leave_Home may be executed after running at least two activities, carried out in any two contexts unrelated to each other. As expected, activity Enter_Home is always carried out after activity Leave_Home, which in turn must be preceded by the execution of at least two more activities, carried out in two different but connected contexts. Again this is sensible, since activity Enter_Home is expected to always follow Leave_Home. As to activity Work, it can be carried out after at least any two other activities, but necessarily in a context in which sensor 'm026' (corresponding to the chair in the studio) has status 'on'. This suggests that the person, for being at work, must necessarily be sitting in the studio chair. Activity Bed_To_Toilet has the same precondition as Meal_Preparation previously described. Activity Resperate may be carried out after any two activities, but requires a very detailed context (possibly due to its being peculiar, or possibly due to only 6 examples being available for it): sensors ‘m006’ (situated on the bedroom door), ‘m008’ (situated at the bedroom exit), ‘m014’ (situated on the chair near the table in the dining room), ‘m018’ (situated at the kitchen entrance), ‘m020’ (situated near the garage entrance), ‘m021’ (situated in the middle of the house in between the various rooms) must all be in status ‘off’; moreover, sensors ‘m022’ (situated in the main corridor), ‘m026’ (situated on the studio chair) and ‘m028’ (situated at the studio entrance) must be involved in any status. Also the context of the activity preceding Resperate has some restrictions: it must involve, in any status, sensors ‘m013’ (situated in the living room), ‘m018’ (situated at the kitchen entrance), ‘m019’ (situated in the kitchen), ‘m020’ (situated in the living zone) and ‘m021’ (situated in the middle of the house).

6 Conclusions

Since many human processes are nowadays very complex, tools that provide automatic support to their accomplishment are welcome. However, the underlying models are too complex for writing and setting up them manually, and even standard machine learning approaches may be unable to infer them. Endowing these models with the capability of specifying conditions that determine whether some tasks are to be carried out or not is a further source of complexity, especially if these conditions may involve sequential relationships.

This paper presented a First-Order Logic approach to learn complex process models extended with conditions, and use the learned models to monitor subsequent process enactment. A real-world experiment was run concerning the daily routines of the user of a smart environment, for predicting his needs and comparing the actual situation with the expected one. Positive results have been obtained, both for efficiency and for effectiveness. In future work, we plan to further extend and improve the expressiveness of the models, and to apply them to different complex domains.

Acknowledgments

This work was partially funded by the Italian PON 2007-2013 project PON02_00563_3489339 ‘Puglia@Service’.

References

[1] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, 1998.

[2] C. R. Anderson, P. Domingos, and D. S. Weld. Relational markov models and their application to adaptive web navigation. In D. Hand, D. Keim, and R. Ng, editors, *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 143–152. ACM Press, 2002.

[3] J.E. Cook and A.L. Wolf. Discovering models of software processes from event-based data. Technical Report CU-CS-819-96, Department of Computer Science, University of Colorado, 1996.

[4] F. Esposito, N. Di Mauro, T.M.A. Basile, and S. Ferilli. Multi-dimensional relational sequence mining. *Fundamenta Informaticae*, 89(1):23–43, 2008.

[5] F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli. Multistrategy theory revision: Induction and abduction in inthelex. *Machine Learning Journal*, 38(1/2):133–156, 2000.

[6] S. Ferilli. Woman: Logic-based workflow learning and management. *IEEE Transaction on Systems, Man and Cybernetics: Systems*, 44:744–756, 2014.

[7] S. Ferilli, T. M.A. Basile, M. Biba, N. Di Mauro, and F. Esposito. A general similarity framework for horn clause logic. *Fundamenta Informaticæ*, 90(1-2):43–46, 2009.

[8] S. Ferilli, B. De Carolis, and D. Redavid. Logic-based incremental process mining in smart environments. In *Recent Trends in Applied Artificial Intelligence*, volume 7906 of *Lecture Notes in Artificial Intelligence*, pages 392–401. Springer, 2013.

[9] S. Ferilli and F. Esposito. A heuristic approach to handling sequential information in incremental ilp. In *AI*IA 2013: Advances in Artificial Intelligence*, Lecture Notes in Artificial Intelligence.

[10] S. Ferilli and F. Esposito. A logic framework for incremental learning of process models. *Fundamenta Informaticae*, 128:413–443, 2013.

[11] B. Gutmann and K. Kersting. Tildecrf: Conditional random fields for logical sequences. In *In Proceedings of the 15th European Conference on Machine Learning (ECML-06*, pages 174–185. Springer, 2006.

[12] J. Herbst and D. Karagiannis. An inductive approach to the acquisition and adaptation of workflow models. In *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, pages 52–57, 1999.

[13] N. Jacobs. Relational sequence learning and user modelling, 2004.

[14] K. Kersting, L. De Raedt, B. Gutmann, A. Karwath, and N. Landwehr. Probabilistic inductive logic programming. chapter Relational sequence learning, pages 28–55. Springer-Verlag, Berlin, Heidelberg, 2008.

[15] K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards discovering structural signatures of protein folds based on logical hidden markov models. Technical Report report00175, Institut f ur Informatik, Universit at Freiburg, 2002, June 13.

[16] S.D. Lee and L. De Raedt. Constraint based mining of first order sequences in seqlog. In *Database Support for Data Mining Applications*, volume 2682 of *Lecture Notes in Computer Science*, pages 155–176. Springer-Verlag, 2004.

[17] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[18] W.M.P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8:21–66, 1998.

[19] A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data. In V. Hoste and G. De Pauw, editors, *Proceedings of the 11th Dutch-Belgian Conference of Machine Learning (Benelearn 2001)*, pages 93–100, 2001.

Visualization for Streaming Networks

Rui Sarmento[1,2], Mário Cordeiro[1], and João Gama[1,2]

[1] LIAAD-INESC TEC, University of Porto
[2] Faculty of Economics, University Porto

Abstract. Sampling from Large Scale Social Networks is a hot topic in recent research. In telecommunications services, there are many networks with millions of nodes and billions of edges. They are complex and difficult to analyze. Sampling, together with vizualization techniques, are required for exploratory data analysis and event detection. Until now, to visualize and analyze the massive network data we would rely on aggregation of communities, k-Core decompositions and matrix feature representations, among others. In social network visualization and analysis the goal is to get more information from the data with the least alienation possible from the actors of the network. Our contribution is to treat the data like a continuous data stream and represent it by sampling the full network. We also propose group visualization and analysis of influential actors in the network by using a Top-K representation of the network data stream.

Keywords: Sampling Large Scale Social Networks; Data Streams; Telecommunication Networks.

1 Motivation

Large network visualization is known to be a hard problem to solve with typical hardware or software sets. It is vulgar to see software and hardware suffer to output networks with more than a few thousands nodes and edges. The software and the observer himself seems to be the main constraints in visualization tasks of large networks. It must be said that even if the software is capable of outputting a network of millions of nodes on the screen it is a very hard task for the observer to gather valuable information from the visual outcome. This document main contribution is our proposal to treat the data as a stream of networked data with Landmark, Sliding Windows and Top-K algorithm implementations to help the observer visualize the network and be able to acquire knowledge from the output. The main goal is to sample the stream by highlighting the Top-K nodes thus providing clear insight on the most active nodes in the network. This document presents a Telecommunications network data case study for application of our methods. The network size is of several millions of nodes and edges. The results were obtained with a vulgar commodity machine.

2 Related Work

2.1 Visualization

The definition of large scale networks regarding number of nodes or edges varies a lot. While researching this field it is usual to see researchers considering a large scale network ranging from something like some dozens of thousands of nodes to millions of nodes and billions of edges. The main goal of any graph visualization technique is to be visually understandable. It is also desirable that the information it outputs to the viewer is sufficiently clear and objective to convey knowledge acquiring. Some studies have been performed to study two types of graph representations, node-link and matrix graph representations like in [13]. Those studies concluded visualization comprehensibility is highly related with the network size (number of nodes) and density (average number of edges per node). Node-link representation is mentioned to have low performance with dense networks and it requires aggregation methods reducing density to increase visual comprehensibility of output. Matrix representation is usually combined with hierarchical aggregation [1]. The hierarchical clustering implies the grouping of nodes but not the ordering of them. The main goal of this representation type is to have a fast clustering algorithm and meaningful clusters of the network. Matrix representation methods can also rely on the reordering of rows and columns of the matrix representation instead of just clustering the nodes and there are several examples of these implementations in [11]. This type of ordered matrix representation might enhance the structure visualization and give more information to the viewer because the data is not only clustered. On the other side this solution is difficult to use with networks of millions of nodes due to the computations needed to reordering of matrix. More recently some innovations were introduced for fast reordering mechanism, data aggregations and GPU-accelerated rendering to deliver solutions with higher scalability [6]. Other solutions rely on controlling the visual density of graph view and limiting the clusterization overlap probability to low levels in [17]. Moreover a new probability based network metric were introduced in [9] to identify potentially interesting or anomalous patterns in the networks. In the next sections we will write our approach by inspecting Top-K actors in the network and also to the case study in hands.

2.2 Top-K Networks

There is some effort by the scientific community to achieve efficient ways of data streams summarization. Regarding streaming networks the exact solution implies the knowledge of all the nodes and edges frequency, therefore this exact solution might be impossible to achieve in large scale networks. The problem of finding the most frequent items in a data stream S of size N is, roughly put, the problem to find the elements e_i whose relative frequency f_i is higher than a user specified support ϕN, with $0 \leq \phi \leq 1$ [7]. Given the space requirements that exact algorithms addressing this problem would need [3], several algorithms were already proposed to find the top-k frequent elements. Generically, there are two different types of approaches: *Counter-based* and *Sketch-based* [16]. *Counter-based* techniques rely on the updates of individual counters for each element in a specific data subset. If there is no counter for some particular element and therefore it is not being monitored some algorithm action is taken. *Counter-based* techniques are said to have fast per-element processing and provable error bounds [16]. *Sketch-based* techniques are different from *Counter-based* topologies because they do not monitor a subset of elements but rely on a frequency estimation for all elements by using bit-maps of counters [16]. Therefore they are more expensive in terms of processing than the *Counter-based* implementations. Moreover *Sketch-based* implementations do not guarantee frequency estimation/approximation errors. Simple *counter-based* algorithms such as *Sticky Sampling* and *Lossy Counting* were proposed in [15], which process the stream in reduced size. Yet, they suffer from keeping a lot of irrelevant counters. *Frequent* [5] keeps only k counters for monitoring k elements, incrementing each element counter when it is observed, and decrementing all counters when a unmonitored element is observed. Zeroed-counted elements are replaced by new unmonitored element. This strategy is similar to the one applied by *Space-Saving* [16], which gives guarantees for the top-m most frequent elements. *Sketch-based* algorithms usually focus on families of hash functions which project the counters into a new space, keeping frequency estimators for all elements. The guarantees are less strict but all elements are monitored. The *CountSketch* algorithm [3] solves the problem with a given success probability, estimating the frequency of the element by finding the median of its representative counters, which implies sorting the counters. Also, *GroupTest* method [4] employs expensive probabilistic calculations to keep the majority elements within a given probability of error. Although generally accurate, its space requirements are large and no information is given about frequencies or ranking.

We adopted the *Space-Saving* algorithm described in [16] throughout our Top-K implementation because it is a memory efficient implementation and guarantees most active users which is our goal.

3 Case Study

The characteristics of the large scale data will be presented in this section. Telecommunication networks generate large amount of continuous data from phone users and network equipment. In this case study we used CDR (call detail records) log files retrieved from equipment distributed geographically. The network data has, on average, 10 million calls (edges in the social network) per day. This represents an average of 6 million of unique users (nodes in the network) per day. Each edge represents a call between A and B phone numbers. We had available 135 days of anonymized data. For each edge/call there is a timestamp information with the date and time with resolution to the second representing the beginning of the call. The volume of data speed ranges from 10 up to 280 calls per second usually around mid-night and mid-day time, respectively.

3.1 Data Description

In a first analysis, we look for the distribution of calls. We started by counting the number of calls from A→B in one day of the data. This operation was done with a MySql database query by selecting pairs of caller and receiver numbers and counting the occurrences of those pairs in the database. The obtained results imply a compressed representation of the original network i.e. without repeated edges. After the previous operation we studied the distribution of the aggregated data and concluded that this representation has a Power Law distribution [2] as can be seen in Fig. 1(left). Therefore we can expect a high number of single calls between some A→B numbers and a low number of many calls between A→B numbers per day.

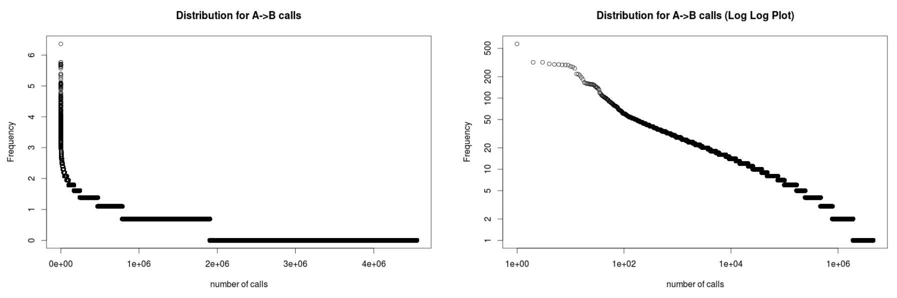


Fig. 1. A→B Calls Distribution (left) and respective Log-Log Plot (right)

We then studied the Log-Log representation of the distribution per day of aggregated data as seen in Fig. 1(right). With this representation we can visualize an approximation to a monomial.

For the received and caller calls distributions of the original data with this same representation method we could also obtain a monomial and we could also conclude both distributions follow a Power Law distribution by using the method to test the Power Law hypothesis in [8].

The previous Figures provides us a visualization of an important data characteristic which is the great amount of isolated calls between some pairs of numbers and a low amount of repeated calls between them. We conclude it is logical to disregard these isolated calls to improve visualization and analysis quality as we will later see in this document with the Top-K visualization method.

3.2 Landmark Windows

Landmark Window Algorithm 1 provides the representation of all the events that occur in the network starting at a specific time stamp, for example 01h48m09s of 1st January 2012.

Algorithm 1 Landmark algorithm Pseudo-Code

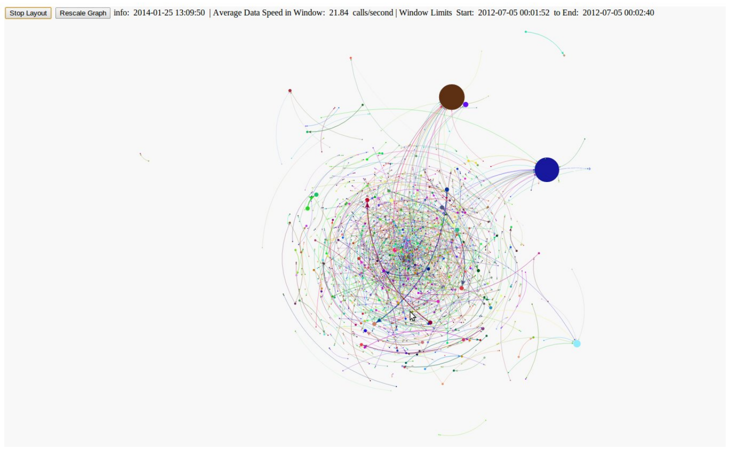
```
Input: $start$, $wsize$, $tinc$    ▷ start timestamp, window size and time increment
Output: $edges$
 1: $R \leftarrow \{\}$                                        ▷ data rows
 2: $E \leftarrow \{\}$                     ▷ edges currently in the graph
 3: $R \leftarrow$ getRowsFromDB ($start$)
 4: $new\_time \leftarrow start$
 5: while ($R <> 0$) do
 6:     for all $edge \in R$ do
 7:         addEdgeToGraph($edge$)
 8:         $E \leftarrow E \bigcup \{edge\}$
 9:     end for
10:     $new\_time \leftarrow new\_time + tinc$
11:     $R \leftarrow$ getRowsFromDB ($new\_time$)
12: end while
13: $edges \leftarrow E$
```

This type of representation is not very useful because it implies a crescent number of events outputted on the screen and comprehensibility lowers as this number reaches and surpasses some thousands of events. This landmark implementation is however useful in other contexts like for example if user network is relatively small and the user wants to check all events in the network. If the user wants to follow the evolution of a large network events the implementation described in the next subsection is better.

3.3 Sliding Windows

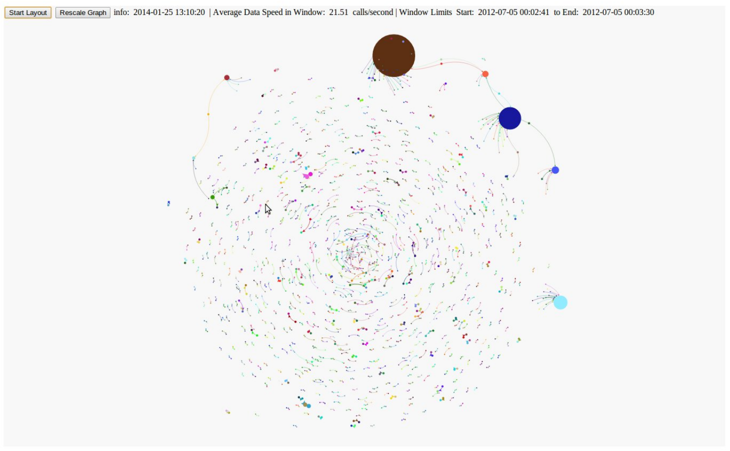
With the need to treat the large data stream we did a dynamic sample representation of the data designated by sliding window. This sliding window is defined as a data structure with fixed number of registered events. Each event is a call between any particular pair of phone numbers. As these events have time stamps the time period between the first call and the last call in the window is easily computed. The input parameters of this algorithm are start date and time and also the maximum number of events/calls the sliding window can have. The SNA model used in this implementation is full network directed since any nodes in the network are outputted to the screen and for the particular window of events[10].

Visually, the example result can be seen in Fig. 2. In this representation several nodes appear bigger and that represents more received/made calls by these particular numbers. This is the representation of a window with 1000 events/calls for a period of time beginning at 00h01m52s and until 00h02m40s. The reader can check the evolution of the network and visually and immediately conclude that the anonymized brown, dark blue and light blue are the nodes with more influence in this window of time.



info: 2014-01-25 13:09:50 | Average Data Speed in Window: 21.84 calls/second | Window Limits Start: 2012-07-05 00:01:52 to End: 2012-07-05 00:02:40

Fig. 2. Visualization of the phone calls using a Sliding Window approach



info: 2014-01-25 13:10:20 | Average Data Speed in Window: 21.51 calls/second | Window Limits Start: 2012-07-05 00:02:41 to End: 2012-07-05 00:03:30

Fig. 3. Visualization of the phone calls using a Sliding Window approach

One can also see the connection between the dark blue node and the brown node being established in the represented window. Fig. 2 also displays the average data speed in the window i.e. the speed was approximately 22 calls per second. This average data speed is calculated regarding number of events/calls in the window of events and the time period between the first event time stamp and the last event time stamp represented in the visualized window. Throughout other experimental conditions for example with windows around mid-day we experienced data speed increases with more calls per second. Considering these data speed changes and after several experiments with window size parameter we concluded that it should not be smaller than approximately 100 events and also not bigger than approximately 1000 events. With the minimum data speed conditions, 100 events represents a window period of around 10 seconds of events. With the maximum data speed and a window of 1000 events it represents around 5 seconds of calls data. Less than 100 events with this data represents changes in the window that are too fast to be visually comprehensible and more than 1000 events represents too much events decreasing visual comprehension of the screen output.

Fig. 3 represents the next window between 00h02m41s and 00h03m30s. From Fig. 2 we can visually check the evolution of the network and immediately conclude that the anonymized brown, dark blue and light blue are the nodes with more influence in this window of 1000 events.

Algorithm 2 Sliding Window algorithm Pseudo-Code

```
Input: $start$, $wsize$, $tinc$        ▷ start timestamp, window size and time increment
Output: $edges$
 1: $R \leftarrow \{\}$                                         ▷ data rows
 2: $E \leftarrow \{\}$                          ▷ edges currently in the graph
 3: $V \leftarrow \{\}$                 ▷ buffer to manage removal of old edges
 4: $R \leftarrow$ getRowsFromDB ($start$)
 5: $new\_time \leftarrow start$
 6: $p \leftarrow \{\}$
 7: while ($R <> 0$) do
 8:     for all $edge \in R$ do
 9:         ADDEDGETOGRAPH($edge$)
10:         $E \leftarrow E \bigcup \{edge\}$
11:         $k \leftarrow 1 + (p \bmod wsize)$
12:         $old\_edge \leftarrow V[k]$
13:         REMOVEEDGEFROMGRAPH($old\_edge$)
14:         $E \leftarrow E \setminus \{old\_edge\}$
15:         $V[k] \leftarrow edge$
16:         $p \leftarrow p + 1$
17:     end for
18:     $new\_time \leftarrow new\_time + tinc$
19:     $R \leftarrow$ getRowsFromDB ($new\_time$)
20: end while
21: $edges \leftarrow E$
```

3.4 Top-K Networks

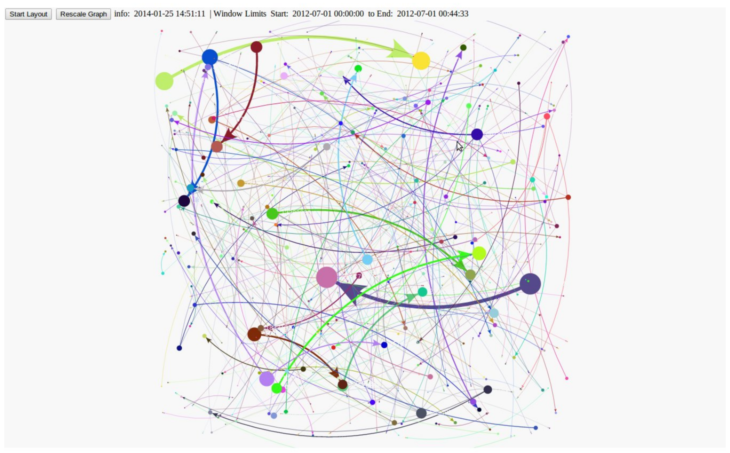
Algorithm 3 represents our version of the Top-K space saving algorithm. The space-saving algorithm is one of the most efficient one-pass algorithms to find the most frequently occurring items in the stream. In our case-study, we are interested in continuously maintain the top-k most active phone users. Activity can be defined as making a call, receiving a call, or communications pairs of users. In this section, we restrict the analysis to the most active calling users.

Algorithm 3 Top-K algorithm Pseudo-Code for made calls inspection

```
Input: $start$, $k\_param$, $tinc$        ▷ start timestamp, k parameter and time increment
Output: $edges$
 1: $R \leftarrow \{\}$                                              ▷ data rows
 2: $E \leftarrow \{\}$                                 ▷ edges currently in the graph
 3: $R \leftarrow$ getRowsFromDB ($start$)
 4: $new\_time \leftarrow start$
 5: while ($R <> 0$) do
 6:     for all $edge \in R$ do
 7:         $before \leftarrow$ getTopKNodes()
 8:         updateTopNodesList($edge$)                ▷ update node list counters
 9:         $after \leftarrow$ getTopKNodes()
10:         $maintained \leftarrow before \bigcap after$
11:         $removed \leftarrow before \setminus maintained$
12:         for all $node \in after$ do                          ▷ add top-k edges
13:             if $node \subset edge$ then
14:                 addEdgeToGraph($edge$)
15:                 $E \leftarrow E \bigcup \{edge\}$
16:             end if
17:         end for
18:         for all $node \in removed$ do         ▷ remove non top-k nodes and edges
19:             removeNodeFromGraph($node$)
20:             for all $edge \in node$ do
21:                 $E \leftarrow E \setminus \{edge\}$
22:             end for
23:         end for
24:     end for
25:     $new\_time \leftarrow new\_time + tinc$
26:     $R \leftarrow$ getRowsFromDB ($new\_time$)
27: end while
28: $edges \leftarrow E$
```

For this representation the input parameters of this algorithm are start date and time and also the maximum number of nodes to be represented (the K parameter). From the inputted start date and time the Top-K implementation would visually output the evolving network of the Top-K actors. The user has also the option to choose to inspect the Top-K network of the numbers that initiate calls, the numbers that receive calls and finally the Top-K representation of the A→B calls. From now on and in this document we define the caller number as the main actor for our Top-K model and we will only provide results and experiments for this situation. Therefore the weight of each actor is related to the number of made calls by each actor i.e. the number of edges representing initiated calls by the focused network phone number.

Fig. 4 represents the output of the Top-100 nodes or phone numbers with more made calls until 00h44m33s. The program started running at midnight of the first day of July 2012 and shown the 100 most active phone numbers in that period.



info: 2014-01-25 14:51:11 | Window Limits Start: 2012-07-01 00:00:00 to End: 2012-07-01 00:44:33

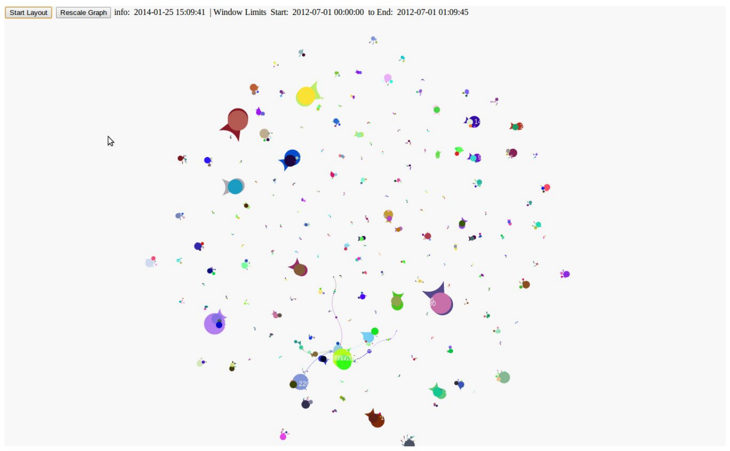
Fig. 4. Top-100 numbers with more made calls and their connections without running the layout algorithm

Fig. 5 represents the output of the Top-100 anonymized nodes or phone numbers with higher number of made calls but now with the layout algorithm running. The output only considers algorithm results until 01h09m45s.

ForceAtlas2 was the chosen layout algorithm. This layout algorithm has some good characteristics [12], [14]. These special ForceAtlas2 characteristics are:

– Continuous layout algorithm, that allows the manipulation of the graph while it is being rendered. It is based on the linear-linear model where the attraction and repulsion are proportional to distance between nodes. The convergence of the graph is considered to be very efficient once that features an unique adaptive convergence speed.
– Proposes summarized settings, focused on what impacts the shape of the graph (scaling, gravity. . .). It is suitable for large graph layout because it features a Barnes Hut optimization (performance drops less with big graphs).

This layout algorithm although being reported to be slow with more than dozens of thousands nodes is perfectly capable of outputting the layout of the used windows sizes throughout all our experiences. As explained before it is



info: 2014-01-25 15:09:41 | Window Limits Start: 2012-07-01 00:00:00 to End: 2012-07-01 01:09:45

Fig. 5. Top-100 numbers with more calls and their connections with layout algorithm running

expected with our data that the windows size do not get much bigger than 1000 events for the Sliding Windows representations and also K parameter lower than 100-200 nodes for the Top-K representation. Numbers much higher than these start to turn the output less understandable and the layout algorithm also becomes slow.

4 Conclusions

This document tries to expose a new type of treatment for Large Scale Telecommunications Networks visualization. With the use of data time stamps we approach the data with a streaming point of view and try to visualize samples of data in a way that is both understandable to the viewer and also allows him/her to gather knowledge from the visual output.

Landmark Windows experiments proved to suffer from the problems we wish to avoid i.e. low visual comprehensibility of the network and even memory issues with the software. This happens when the number of nodes and edges exceeds dozens of thousands of nodes. With our data this number of nodes represented in the screen typically corresponds to a time period of just a few minutes. Sliding windows were used as a way to continuously check for the full network events. Sliding Windows allow us to continuously inspect temporal evolution of the networks. The Top-K implementation is a very good approach to our data presenting a Power Law distribution for calls. This allows us to focus on the influential individuals and discard isolated calls which are the majority of calls in our data. Finally we conclude that our method for evolving networks visualization, specially with Sliding Windows or the Top-K model is a light method to visualize massive networks. The use of a vulgar commodity machine made possible to simulate a data stream and achieve visualization results very proximate to the node-link level. This means we tried avoiding other types of representations previously mentioned in this document's related work. These other types however use hierarchical aggregation of features, for example node communities that we could also add i.e. simultaneously make community detection of the network and output the network with added information to the node-level. This is true for communities, centrality measures like betweeness or closeness centrality could also be added to the node information complementing its visualization on the screen.

The goal and future work is to use this kind of real time data streaming leveraging telecommunication systems and being able to visualise the evolving network in real time. This can lead to applicable uses for fraud detection by inspection of Top-K users in the network or commercial purposes by detecting central actors in the network for example. If more information was available in the data it could also be added to the visual output. A simple mouse pointer over the node and the additional node information could be interactively checked by the system user. Future work also includes testing the models with time decay factors to be possible to use for example the Landmark model, giving more importance to recent data and disregarding old data.

Acknowledgments

This work was supported by Sibila and Smartgrids research projects (NORTE-07-0124-FEDER-000056/59), financed by North Portugal Regional Operational Programme (ON.2 O Novo Norte), under the National Strategic Reference Framework (NSRF), through the Development Fund (ERDF), and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT), and by European Commission through the project MAESTRA (Grant number ICT-2013-612944). The authors also acknowledge the financial support given by the project number 18450 through the "SI I&DT Individual" program by QREN and delivered to WeDo Business Assurance. Finally the authors acknowledge the reviewers for their constructive reviews on this document.

References

1. James Abello and Frank van Ham. Matrix zoom: A visual interface to semi-external graphs. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '04, pages 183–190, Washington, DC, USA, 2004. IEEE Computer Society.
2. A. L. Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, (435):207–211, 2005.
3. Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 693–703, London, UK, UK, 2002. Springer-Verlag.

4. Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. 2003.
5. Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. Frequency estimation of internet packet streams with limited space. In *Algorithms-ESA 2002*, pages 348–360. Springer, 2002.
6. Niklas Elmqvist, Thanh-Nghi Do, Howard Goodell, Nathalie Henry, and Jean-Daniel Fekete. ZAME: Interactive Large-Scale Graph Visualization. In IEEE Press, editor, *IEEE Pacific Visualization Symposium 2008*, pages 215–222, Kyoto, Japon, 2008. IEEE.
7. Joao Gama. *Knowledge Discovery from Data Streams.* Chapman & Hall/CRC, 1st edition, 2010.
8. Colin S Gillespie. *Fitting heavy tailed distributions: the poweRlaw package*, 2014. R package version 0.20.5.
9. Frank Ham, Hans-Jörg Schulz, and Joan M. Dimicco. Honeycomb: Visual analysis of large scale social networks. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part II*, INTERACT '09, pages 429–442, Berlin, Heidelberg, 2009. Springer-Verlag.
10. Robert A. Hanneman and Mark Riddle. *Introduction to Social Network Methods.* University of California, Riverside, Riverside, CA, USA, 2005.
11. Nathalie Henry and Jean daniel Fekete. Matrixexplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12:677–684, 2006.
12. M. Jacomy. Forceatlas2, the new version of our home-brew layout., 2013. [Online; accessed 21-Dec-2013].
13. Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, pages 1–5, New York, NY, USA, 2006. ACM.
14. T. Venturini M. Jacomy, S. Heymann and M. Bastian. Forceatlas2, a graph layout algorithm for handy network visualization, 2011. [Online; accessed 29-Dec-2013].
15. G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.
16. Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th International Conference on Database Theory*, ICDT'05, pages 398–412, Berlin, Heidelberg, 2005. Springer-Verlag.
17. Lei Shi, Nan Cao, Shixia Liu, Weihong Qian, Li Tan, Guodong Wang, Jimeng Sun, and Ching-Yung Lin. Himap: Adaptive visualization of large-scale online social networks. In Peter Eades, Thomas Ertl, and Han-Wei Shen, editors, *PacificVis*, pages 41–48. IEEE Computer Society, 2009.

Dependency Detection in Interval-based Event Streams

Vasile-Marian Scuturici[1], Marc Plantevit[2], and Céline Robardet[1]

[1]Université de Lyon, CNRS, INSA-Lyon, LIRIS UMR5205,
F-69621 Villeurbanne, France
[2]Université de Lyon, CNRS, Univ. Lyon1, LIRIS UMR5205,
F-69622 Villeurbanne, France

Abstract. We present a new approach to mine dependencies between streams of interval-based events that links two events if they occur in a similar manner, one being often followed by the other one in the data. The proposed technique is robust to temporal variability of events and determines the most appropriate time intervals whose validity is assessed by a χ^2 test. TEDDY algorithm prunes the search space while certifying the discovery of all valid and significant temporal dependencies.

1 Introduction

The recent breakthroughs in sensor technology have given users the ability to monitor many events in real time producing multiple heterogeneous data streams. This highly innovative context has been a fruitful source of motivation for the development of many data stream management and analysis techniques that extend classical pattern mining techniques to be able to mine the data faster than the data generation process. In this paper, events are characterized by a set of intervals in which they occur and we aim at identifying temporal dependencies between them. Two events are linked if the intervals of one are repeatedly followed by the intervals of the other one. Considering time intervals makes it possible to improve existing time-point based approaches by (1) better handling events that are rare but occur for a long period of time; (2) being more robust to the temporal variability of events; (3) allowing the discovery of sophisticated relations based on Allen's algebra [10]. Our interval-based approach also determines the most appropriate time-delay intervals that may exist between them. Valid and significant temporal dependencies are mined: The strength of the dependency is evaluated by the proportion of time where the two events intersect and its significance is assessed by a χ^2 test. As several intervals may redundantly describe the same dependency, the approach retrieves only the few most specific ones with respect to a dominance relationship. Discovering all valid and significant temporal dependencies is challenging since, for every couple of events, all possible time-delay intervals have to be considered. Therefore, we propose an efficient algorithm TEDDY, TEmporal Dependency DiscoverY, that benefits from different properties in order to prune the search space while certifying the completeness of the extraction.

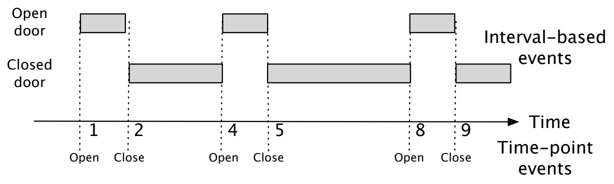


Fig. 1. An example.

2 Temporal dependencies

Data streams are generally considered as temporal sequences of time-point events, $S =< (a, t) >$, that is to say sequences of couples made of a nominal symbol $a \in \mathcal{A}$, and a time stamp $t \in T_s$, with T_s the discrete time of observation. For example on Fig. 1, $\mathcal{A} = \{open, close\}$ and the time-point events are $< (open, 1), (close, 2), \cdots, (close, 9) >$. But, in many application domains, this is the time interval between time-point events that conveys the most valuable information. For example, the time intervals during which a door is open may be in temporal dependency with the detection of a moving object by a camera. Therefore, it can be interesting to examine the intervals associated to these events. A point-based event sequence S is turned into as many *interval-based event sequences* as there are symbols $a \in \mathcal{A}$. The resulting interval-based sequences are denoted by capital letter A. Thus, the interval-based sequence associated to the event a is denoted A and is defined by:

$$A =< [t_i, t_{i+1}) \mid t_i, t_{i+1} \in T_s > \text{ where } \forall t \in ([t_i, t_{i+1}) \cap T_s)\,,\ (a, t) \in S$$

Following the example on Fig. 1, the interval set associated to the event *open* is *Open door* $=< [1, 2),\ [4, 5),\ [8, 9) >$ and the one associated to *close* is *Closed door* $=< [2, 4),\ [5, 8) >$. The significance of an interval-based event, called *event* hereafter, is evaluated by the sum of the lengths of its intervals: $\mathbf{len}(A) = \sum_{[t_i,t_{i+1})\in A}(t_{i+1} - t_i)$. On Fig. 1, $\mathbf{len}(Open\ door) = 3$

The dependency of two events A and B is evaluated on the basis of the intersection of their intervals: $\mathbf{len}(A \cap B) = \mathbf{len}(< [t_i, t_{i+1}) \cap [t_j, t_{j+1}) >)$ with $[t_i, t_{i+1}) \in A$ and $[t_j, t_{j+1}) \in B$). However, two events A and B can be in temporal dependency $A \rightarrow B$ while not being synchronous. It happends when B is time-delayed with respect to A. To capture such dependencies the intervals of B may undergo some transformations so as to better coincide with the intervals of A: (1) B can be shifted of β time units so as to maximize its intersection with A (the two end-points of the intervals are advances of β time units), and (2) B can be slightly extended so as to make the temporal dependency measure more robust to the inherent variability of the data (the first end-point is advanced of α time units and the second end-point is advanced of β time units, with α slightly greater than β): $B^{[\alpha,\beta]} =< [t_j - \alpha, t_{j+1} - \beta) >$ with $[t_j, t_{j+1}) \in B$ and $\alpha \geq \beta \geq 0$.

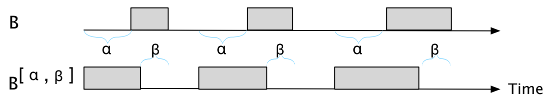


Fig. 2. Example of interval set shifts

2.1 Temporal dependency assessment

Given a shifting interval $[\alpha, \beta]$, the temporal dependency of $A \xrightarrow{[\alpha,\beta]} B$ is evaluated by the proportion of time where the two events simultaneously occur over the length of A:

$$\mathbf{conf}(A \xrightarrow{[\alpha,\beta]} B) = \frac{\mathbf{len}(A \cap B^{[\alpha,\beta]})}{\mathbf{len}(A)}$$

We can observe that $\mathbf{conf}(A \xrightarrow{[\alpha,\beta]} B)$ is equal to 1 iff each interval of A is included in an interval of $B^{[\alpha,\beta]}$. To statistically assess the value of $\mathbf{conf}(A \xrightarrow{[\alpha,\beta]} B)$, we propose to perform a Pearson's chi-squared test of independence [8]. The test determines whether or not the occurrences of A and $B^{[\alpha,\beta]}$ are statistically independent over the period of observation T defined by $T = [t_{begin}, t_{end})$ with

$$t_{begin} = \min\{\min_{[t_i,t_{i+1})\in A} t_i, \min_{[t_j,t_{j+1})\in B} t_j\} \text{ and } t_{end} = \max\{\max_{[t_i,t_{i+1})\in A} t_{i+1}, \max_{[t_j,t_{j+1})\in B} t_{j+1}\}$$

A given time point of T might belong or not to an interval of A. These two possible outcomes are denoted $\mathbf{A}$ and $\overline{\mathbf{A}}$. Table 1 (top) is the contingency table O that crosses the observed outcomes of A and $B^{[\alpha,\beta]}$. The null hypothesis states

	$\mathbf{B}^{[\alpha,\beta]}$	$\overline{\mathbf{B}^{[\alpha,\beta]}}$
$\mathbf{A}$	$\mathbf{len}(A \cap B^{[\alpha,\beta]})$	$\mathbf{len}(A) - \mathbf{len}(A \cap B^{[\alpha,\beta]})$
$\overline{\mathbf{A}}$	$\mathbf{len}(B^{[\alpha,\beta]}) - \mathbf{len}(A \cap B^{[\alpha,\beta]})$	$\mathbf{len}(T) - \mathbf{len}(A) - \mathbf{len}(B^{[\alpha,\beta]}) + \mathbf{len}(A \cap B^{[\alpha,\beta]})$

Matrix O of observations.

	$\mathbf{B}^{[\alpha,\beta]}$	$\overline{\mathbf{B}^{[\alpha,\beta]}}$
$\mathbf{A}$	$\frac{\mathbf{len}(B^{[\alpha,\beta]}) \times \mathbf{len}(A)}{\mathbf{len}(T)}$	$\frac{\left(\mathbf{len}(T) - \mathbf{len}(B^{[\alpha,\beta]})\right) \times \mathbf{len}(A)}{\mathbf{len}(T)}$
$\overline{\mathbf{A}}$	$\frac{\mathbf{len}(B^{[\alpha,\beta]}) \times (\mathbf{len}(T) - \mathbf{len}(A))}{\mathbf{len}(T)}$	$\frac{\left(\mathbf{len}(T) - \mathbf{len}(B^{[\alpha,\beta]})\right) \times (T - \mathbf{len}(A))}{\mathbf{len}(T)}$

Matrix E of expected outcomes under the null hypothesis.

Table 1. χ^2 statistic computation.

that the occurrences of $\mathbf{A}$ and $\mathbf{B}^{[\alpha,\beta]}$ are statistically independent: If we suppose that $\mathbf{A}$ occurs uniformly over T, there are $\frac{\mathbf{len}(A)}{\mathbf{len}(T)}$ chances that event $\mathbf{B}^{[\alpha,\beta]}$ occurs at the same time. As $\mathbf{B}^{[\alpha,\beta]}$ occurs during $\mathbf{len}(B^{[\alpha,\beta]})$ time stamps, the expected number that $\mathbf{B}^{[\alpha,\beta]}$ occurs simultaneously with A under the null hypothesis is $\frac{\mathbf{len}(B^{[\alpha,\beta]})\times\mathbf{len}(A)}{\mathbf{len}(T)}$. The three other outcomes under the null hypothesis are constructed on the same principle. All these expected outcomes E are given in table 1 (bottom). The value of the statistical test is

$$\begin{aligned} X^2 &= \sum_{i=1}^{2}\sum_{j=1}^{2}\frac{(O_{ij}-E_{ij})^2}{E_{ij}} \\ &= \frac{\mathbf{len}(T)\left(\mathbf{len}(T)\,\mathbf{len}\left(A\cap B^{[\alpha,\beta]}\right)-\mathbf{len}(A)\mathbf{len}(B^{[\alpha,\beta]})\right)^2}{\mathbf{len}(A)\mathbf{len}(B^{[\alpha,\beta]})(\mathbf{len}(T)-\mathbf{len}(A))(\mathbf{len}(T)-\mathbf{len}(B^{[\alpha,\beta]}))} \end{aligned} \tag{1}$$

The null distribution of the statistic is approximated by the χ^2 distribution with 1 degree of freedom, and for a significant level of 5%, the critical value is equal to $\chi^2_{0.05}=3.84$. Consequently, X^2 has to be greater than 3.84 to establish that the intersection is sufficiently large not to be due to chance. From equation (1) we derive the following quadratic equation in $\mathbf{len}\left(A\cap B^{[\alpha,\beta]}\right)$:

$$\begin{aligned} &\left(\mathbf{len}(T)\,\mathbf{len}\left(A\cap B^{[\alpha,\beta]}\right)-\mathbf{len}(A)\mathbf{len}(B^{[\alpha,\beta]})\right)^2\geq \\ &\frac{3.84}{\mathbf{len}(T)}\mathbf{len}(A)\mathbf{len}(B^{[\alpha,\beta]})(\mathbf{len}(T)-\mathbf{len}(A))(\mathbf{len}(T)-\mathbf{len}(B^{[\alpha,\beta]})) \end{aligned}$$

which is satisfied iff $0\leq\mathbf{len}\left(A\cap B^{[\alpha,\beta]}\right)\leq\cap_1$ or $\mathbf{len}(T)\geq\mathbf{len}\left(A\cap B^{[\alpha,\beta]}\right)\geq\cap_2$, $\cap_1$ and $\cap_2$ being the roots of this equation. Intersection values that range between 0 and $\cap_1$ are much smaller than the ones expected under the null hypothesis. Such values can be used to detect anomalies, but, in the following we focus on the intersection values that are unexpectedly high. Therefore, we conclude that a temporal dependency $A\xrightarrow{[\alpha,\beta]}B$ is valid iff $\mathbf{conf}(A\xrightarrow{[\alpha,\beta]}B)\geq\frac{\cap_2}{\mathbf{len}(A)}$. As the χ^2 test only works well when the dataset is large enough, we use the conventional rule of thumb [8] that enforces all the expected numbers (cells in Table 1 (bottom)) to be greater than 5.

2.2 Significant temporal dependencies selection

For two events in temporal dependency, a huge number of shifting intervals $[\alpha,\beta]$ may exist that result in valid temporal dependencies. These intervals may describe distinct temporal dependencies (e.g., different paths may exist between two motion captors), but they can also depict the same phenomenon several times. Redundancy mainly relies on confidence monotonicity:

Property 1 (Confidence monotonicity). Let A and B be two events and $[\alpha_1,\beta_1]$, $[\alpha_2,\beta_2]$ be two shifting intervals. If $[\alpha_1,\beta_1]\subseteq[\alpha_2,\beta_2]$, then $\mathbf{conf}(A\xrightarrow{[\alpha_1,\beta_1]}B)\leq\mathbf{conf}(A\xrightarrow{[\alpha_2,\beta_2]}B)$.

Proof. $[\alpha_1,\beta_1]\subseteq[\alpha_2,\beta_2]$ implies that $B^{[\alpha_1,\beta_1]}\subseteq B^{[\alpha_2,\beta_2]}$ and $\mathbf{len}(B^{[\alpha_1,\beta_1]}\cap A)\leq\mathbf{len}(B^{[\alpha_2,\beta_2]}\cap A)$. As a result, $\mathbf{conf}(A\xrightarrow{[\alpha_1,\beta_1]}B)\leq\mathbf{conf}(A\xrightarrow{[\alpha_2,\beta_2]}B)$. $\square$

To best describe the temporal dependencies of two events while avoiding the pattern redundancy, we consider the intervals that have (1) a high confidence value and (2) be as specific as possible with respect to the inclusion relation. This leads to the following definition of the *dominance* relationship:

Definition 1 (Dominance relationship). *We say that* $A \xrightarrow{[\alpha_1,\beta_1]} B$ *dominates* $A \xrightarrow{[\alpha_2,\beta_2]} B$*, denoted* $\preceq$*, iff* $[\alpha_1, \beta_1] \subseteq [\alpha_2, \beta_2]$ *and*

$$1 - \frac{\mathbf{conf}(A \xrightarrow{[\alpha_1,\beta_1]} B)}{\mathbf{conf}(A \xrightarrow{[\alpha_2,\beta_2]} B)} < 1 - \frac{\mathbf{len}(B^{[\alpha_1,\beta_1]})}{\boldsymbol{\imath}(B^{[\alpha_2,\beta_2]})} \tag{2}$$

The rationale behind this definition is that when $[\alpha_1, \beta_1]$ dominates $[\alpha_2, \beta_2]$, the loss of the confidence measure of $[\alpha_1, \beta_1]$ is less than the reduction of its interval set length and thus $B^{[\alpha_2,\beta_2] \setminus [\alpha_1,\beta_1]} \cap A$ is almost empty. Indeed, if the reduction of the interval length of $B^{[\alpha,\beta]}$ is uniformly distributed over $[t_{begin}, t_{end}]$, then the length of its intersection with A will be reduced in the same proportion. But, if the reduction mainly occurs when A does not occur, then the length of its intersection with A decreases less, as stated by equation (2).

This dominance relationship makes it possible to refine an interval while controlling the loss of the confidence measure. If an interval reduction leads to a significant loss, then the refinement process has to be stopped, since the portion of A non covered by the interval will not be subsequently either. Therefore, significant temporal dependencies are the most specific temporal dependencies that dominate all their supersets:

Definition 2 (Significant temporal dependencies). *For two events* A *and* B*, let* Σ *be the set of temporal dependencies* $d_{[\alpha,\beta]} = A \xrightarrow{[\alpha,\beta]} B$ *such that (i)* $d_{[\alpha,\beta]}$ *dominates all of its supersets, and (ii) every superset of* $d_{[\alpha,\beta]}$ *dominates its supersets as well:*

$$\begin{aligned}\Sigma = \{d_{[\alpha_1,\beta_1]} \mid \forall\, [\alpha_2,\beta_2] \textit{ such that } [\alpha_1,\beta_1] \subseteq [\alpha_2,\beta_2],\ d_{[\alpha_1,\beta_1]} \preceq d_{[\alpha_2,\beta_2]} \\ \textit{and } \forall\, [\alpha_3,\beta_3] \textit{ such that } [\alpha_2,\beta_2] \subseteq [\alpha_3,\beta_3],\ d_{[\alpha_2,\beta_2]} \preceq d_{[\alpha_3,\beta_3]}\}\end{aligned}$$

Temporal dependencies that belong to the positive border of $(\Sigma, \preceq)$ *are said to be significant.*

Property 2 (Σ-belonging monotonicity). Let $[\alpha_1, \beta_1] \subseteq [\alpha_2, \beta_2]$. From definition 2, we can derived that, if $d_{[\alpha_1,\beta_1]}$ belongs to Σ, then $d_{[\alpha_2,\beta_2]} \in \Sigma$.

3 Efficient Temporal Dependencies Discovery

Discovering temporal dependencies is time-consuming for large volumes of data. Considering that there is no meaning to look for temporal dependencies with large time lag, we restrict the search of shifting intervals $[\alpha, \beta]$ in $[t_{min}, t_{max}]$ set by the end-user. A naive algorithm, that looks for dependencies between two events A and B, will explore all possible time shift intervals included in $[t_{min}, t_{max}]$, whose number is in $\Theta((t_{max} - t_{min})^2)$. For each interval, it computes its confidence value in $\Theta(\#I)$, where $\#I$ is the number of intervals of A or B. Such an algorithm has to be executed with a relatively high frequency over data stream batches of length T. Our proposed algorithm TEDDY, TEmporal Dependency DiscoverY, (1) takes advantage of the monotonic property of the confidence measure, as stated in property 1; (2) exploits an upper bound on the confidence measure, whose complexity is $O(1)$; (3) explores the search space using a level-wise approach in order to discover significant temporal dependencies while computing the confidence value of each interval at most once.

Algorithm 1 `TEDDY`

```
Require: $IS$ a set of interval-based sequences, $[t_{begin}, t_{end})$, and $[t_{min}, t_{max}]$.
Ensure: All significant temporal dependencies over $IS$.
 1: for all $A \in IS$ do
 2:   for all $B \in IS$ do
 3:     $Border \leftarrow \emptyset$
 4:     $\text{Cand}_0 \leftarrow [t_{min}, t_{max}]$
 5:     $d \leftarrow 0$
 6:     while $\text{Cand}_d \neq \emptyset$ do
 7:       $\text{Prom}_d \leftarrow$ Pruning_based_on_confidence($\text{Cand}_d$)
 8:       $[\Sigma_d, Border] \leftarrow$ Pruning_based_on_dominance($\text{Prom}_d$, $Border$)
 9:       $\text{Cand}_{d+1} \leftarrow$ Candidate_generation($\Sigma_d$)
10:       $d \leftarrow d + 1$
11:     end while
12:     $\text{Significant}_{A \to B} \leftarrow$ Compute_valid_and_significant_TD($Border$)
13:   end for
14: end for
15: return $\bigcup_{A,B} \text{Significant}_{A \to B}$
```

TEDDY is sketched in Algorithm 1. For every pair of events, it explores the temporal dependencies in a breadth-first approach. The inclusion operation over time shift intervals defines a semi-lattice, where intervals at given depth d have the length $t_{max} - t_{min} - d$ and are denoted Cand_d. Line 7, Prom_d is computed as the restriction of Cand_d to the dependencies whose confidence value is greater than the lower bound defined in property 4. If a dependency dominates its two ancestors, then it is a promising dominant candidate and thus belongs to Σ_d (line 8). As such, it is added to the *Border* set whereas its ancestors are removed. Line 9, $d+1$-depth candidates are generated if their d-depth ancestors belongs to Σ_d. Line 12 processes *Border* to only extract valid and significant dependencies. This four most important steps are detailed below.

Candidate time shifts generation: As stated in property 1, the confidence measure increases monotonically with time shift interval inclusion. In addition, property 2 stipulates that Σ-belonging is also a monotonic property. So, to prune the search space made of temporal dependencies that are not valid or not significant, the interval semilattice is traversed from the largest interval down to the singletons. If a time shift interval is not valid or does not dominate one of its direct ancestors, then none of the intervals included in it can be a solution. As each interval at depth $d+1$ is included in at most two intervals at depth d, we generate $d+1$-depth candidates by intersecting two elements of Σ_d.

Pruning-based on confidence measure: In order to avoid the computation of the confidence values of unpromising dependencies, we consider the following property, that bounds the difference of confidence between two time shift intervals:

Property 3 (Bounds on confidence). Let A and B be two events, and $[\alpha_1, \beta_1]$ and $[\alpha_2, \beta_2]$ be two time shift intervals:

$$|\mathbf{conf}(A \xrightarrow{[\alpha_1,\beta_1]} B) - \mathbf{conf}(A \xrightarrow{[\alpha_2,\beta_2]} B)| \leq \frac{(|\alpha_1 - \alpha_2| + |\beta_1 - \beta_2|) \times \#B}{\mathbf{len}(A)}$$

where $\#B$ represents the number of intervals in B.

Proof. By shifting an interval $[t_j - \alpha_1, t_{j+1} - \beta_1] \in B^{[\alpha_1,\beta_1]}$ with $[\alpha_2 - \alpha_1, \beta_2 - \beta_1]$, the length of the resulting interval may win or lose a maximum of $(|\alpha_1 - \alpha_2| + |\beta_1 - \beta_2|)$ time units. By multiplying this quantity by the number of intervals in B, the result follows. $\square$

Furthermore, as stated by the χ^2-based threshold, valid temporal dependencies have a confidence value greater than

$$\text{MinConfidence}\,(L(\alpha,\beta)) \equiv \frac{\lambda L(\alpha,\beta) + \sqrt{\frac{3.84}{T}\lambda(T-\lambda)L(\alpha,\beta)(T - L(\alpha,\beta))}}{\lambda T}$$

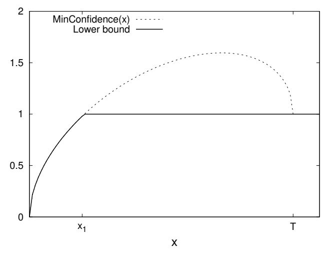
where $L(\alpha,\beta) = \mathbf{len}(B^{[\alpha,\beta]})$ and $\lambda = \mathbf{len}(A)$. Property 4 provides a lower bound on MinConfidence $(L(\alpha,\beta))$:

Property 4 (Lower bound on MinConfidence $(L(\alpha,\beta))$).

$$\text{MinConfidence}\,(L(\alpha,\beta)) \geq \min\,(1, \text{MinConfidence}\,(L(0,0)))$$

Proof. $L(\alpha,\beta)\,(T - L(\alpha,\beta))$ is a quadratic function which vanishes at $L(\alpha,\beta) = 0$ and $L(\alpha,\beta) = T$. Therefore, MinConfidence $(L(\alpha,\beta))$ first increases and then decreases over $[0,T]$ with MinConfidence $(0) = 0$ and MinConfidence $(T) = 1$. Let $x_1 < T$ be such that MinConfidence $(x_1) = 1$. We can observe that MinConfidence (x) increases over $[0, x_1]$ (see figure in the following paragraph). As $L(\alpha,\beta) \geq L(\alpha,\alpha) = L(0,0)$, we have:
MinConfidence $(L(\alpha,\beta)) \geq \min\,(1, \text{MinConfidence}\,(L(0,0)))$. $\square$

$\mathbf{conf}(A \xrightarrow{[\alpha,\beta]} B)$ is upper bounded by 1, therefore if MinConfidence > 1, there is no valid temporal dependency. Algorithm 2 details the evaluation of the confidence measure. The confidence value of the first candidate is computed (line 4). Then, the confidence value of the following candidates is estimated based on Property 3 (line 7). If the upper-bound (`lastConf + maxGain`) of the confidence value of a candidate is lower than MinConfidence $(L(0,0))$ (`boundMinConfidence`, estimated thanks to property 4), then the candidate cannot be valid. Otherwise, its exact confidence is evaluated (line 10) and, if it is greater than `boundMinConfidence` (line 11), the candidate is considered as a promising valid temporal dependency. Notice that the confidence measure is stored for future needs (line 12). This confidence value is used as a new reference for further `maxGain` evaluations, since `maxGain` tends to decrease when evaluated on distant intervals in Cand.



Algorithm 2 `Pruning_based_on_confidence`

```
Require: Cand, an ordered list of candidate intervals, $\#B$ and $\mathbf{len}(A)$.
Ensure: Prom, the set of promising valid dependencies and their confidence values.
 1: Prom ← $\emptyset$
 2: $k \leftarrow 0$
 3: $[\alpha,\beta]$ ←Cand[$k$]
 4: lastConf← conf($A \xrightarrow{[\alpha,\beta]} B$)
 5: while $k < \#Cand$ do
 6:    $[\alpha_k,\beta_k]$ ←Cand[$k$]
 7:    maxGain← $(|\alpha-\alpha_k| + |\beta-\beta_k|) \times \frac{\#B}{\mathbf{len}(A)}$
 8:    if (lastConf + maxGain ≥ boundMinConfidence then
 9:       $[\alpha,\beta]$ ←Cand[$k$]
10:       lastConf← conf($A \xrightarrow{[\alpha,\beta]} B$)
11:       if (lastConf ≥ boundMinConfidence then
12:          Cand[$k$].confidence← lastConf
13:          Prom ← Prom ∪ $Cand[k]$
14:       end if
15:    end if
16:    $k \leftarrow k+1$
17: end while
18: return Prom
```

Pruning-based on dominance relationships: It consists simply in evaluating whether each promising candidate satisfies equation (2) for its direct ancestors. If so, it is added to the *Border* set whereas its ancestors are removed.

Dataset	# Events	Duration	Avg events
SYNT02	85,806	3 hours	2
SYNT04	173,645	3 hours	4
SYNT16	696,677	3 hours	18

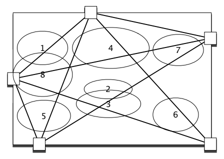


Fig. 3. Dataset characteristics (left). Synthetic testbed (right).

Identification of valid and significant dependencies: Finally, TEDDY checks whether the dependencies of *Border* are valid: It removes any dependencies that are more general than another one and recursively considers its direct ancestors. If Prom is implemented as an interval tree, evaluating that a temporal dependency is the most specific among n elements can be done in $O(\log(n))$. Finding all the dependencies of Prom that are more general than $d_{[\alpha,\beta]}$ can be done in $O(\min(n, k\log(n)))$ where k is the number of output dependencies [4].

4 Experimental Study

This section reports experimental results that illustrate the performance of TEDDY. We use a multi-camera test bed that makes possible to specify the number of generated events and their frequencies. All experiments were performed on a 8 GB RAM computer with a octo-core processor cadenced at 3 GHz, running Windows 7. TEDDY algorithm is implemented in standard C++.

We built a simulator of a sensor surveillance network that consists in the simulation of 8 video cameras. Each camera captures the images of an elliptical area as described in Fig. 3 (right). The simulation consists in moving objects along eight predefined rectilinear paths. To control the number of events occurring per unit of time, objects are generated according to a Poisson distribution. The area covered by each camera is divided into 27 subareas that make in total 216 data streams. An interval-based event *"object detected"* corresponds to objects located in the associated subarea. We generate three datasets which differ from the average number of events per minute and sequence (see Fig. 3 left).

We study the behavior of TEDDY with respect to various parameters: the frequency of events, the period of observation T and t_{max}. In all the experiments, t_{min} is set to 0. Besides, we examine the impact of the constraints that define valid and significant temporal dependencies on the search space size as well as on the execution time. To this end, the four following configurations of algorithm 1 are studied: (1) **WP** (without pruning): lines 7 and 8 are removed and all possible temporal dependencies are considered; (2) **Chi2** (χ^2-based pruning): line 8 is removed and only the constraint on the confidence measure is pushed aside to reduce the search space; (3) **Gradient** (dominance-based pruning): line 7 is removed and only the dominance constraint makes it possible to discard unpromising dependencies; (4) **TEDDY**: both constraints are fully exploited as presented in algorithm 1.

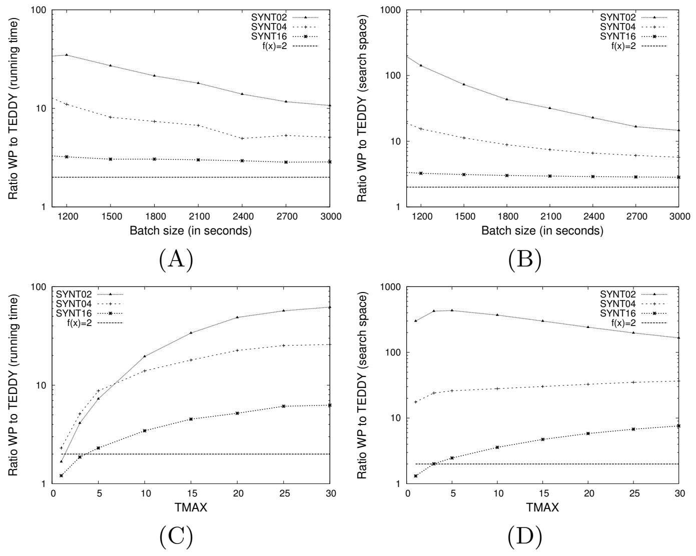


Fig. 4. Ratio of WP to TEDDY. W.r.t. T ($t_{max} = 10$): Runtime (A) and search space size (B); W.r.t. t_{max} ($T = 900$): Runtime (C) and search space size (D).

We first study the performance of TEDDY in comparison with WP. This algorithm considers all dependencies and removes, in a post-treatment, the non valid or non significant ones. For these experiments, we do not take into account the execution time required by this post-processing step. Fig. 4 depicts the behavior of TEDDY when T and t_{max} vary by evaluating the running time and search space size ratios of WP to TEDDY. Each value is averaged over all the sequences of the same size. In most cases, TEDDY is at least twice as fast as WP. The ratio of the execution time increases with t_{max} since the number of intervals is quadratic in $t_{max} - t_{min}$ and TEDDY can prune a large part of them early on. On the contrary, when T increases, the ratio tends to decrease since the number of intervals of each event tends to increase and TEDDY do not prune the search space as much. Indeed, MAXGAIN increases linearly with $\#B$ and the condition at line 8 of algorithm 2 tends to be always true which implies that the time interval cannot be pruned. Furthermore, we can notice that the denser the datasets, the lower the ratios are. The number of extracted dependencies increases with the dataset density as well as the size of the search space.

Fig. 5 shows the proportion of the search space explored by TEDDY. Among the pruned candidates, we make a distinction between those removed thanks to the chi2-based or the gradient-based constraint. A first observation is that the number of candidates avoided thanks to the two constraints is much higher than the number of dependencies considered by TEDDY, except when the dataset is very dense and t_{max} very small. The gradient constraint is even more efficient when the dataset density increases or the values of T and t_{max} grow. While T increases, the number of candidates avoided with gradient-based constraint

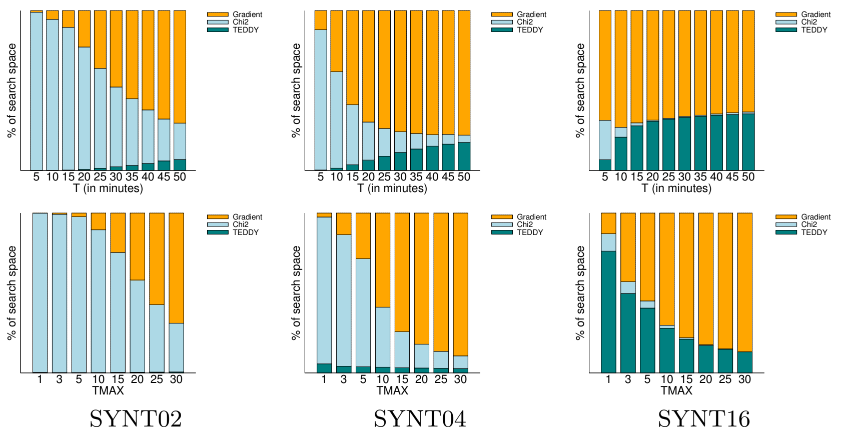


Fig. 5. Constraint impact on the search space size w.r.t. T (top) and t_{max} (bottom).

increases or remains stable whatever the dataset density. This pruning criterion becomes even more effective when t_{max} increases. The larger the length of a pruned interval, the greater the size of the pruned search space. Indeed, if an interval $[\alpha, \beta]$ does not dominate one of its direct ancestors, it is pruned by the gradient-based constraint as well as $\frac{(\beta-\alpha)\times(\beta-\alpha+1)}{2} - 1$ other candidates. Beside, chi2-based pruning tends to be less efficient when t_{max} and/or T increase.

Similarly to what has been observed from Fig. 5, Fig. 6 shows that TEDDY benefits from the two pruning techniques. Notice that the execution time is always much lower than T length (at least 200 times) and thus, the temporal dependencies computation is faster than the data acquisition process.

5 Related work

The proposed approch is related to time series research area where algorithms are devised for measuring the similarity between time series pairs [9]. Most of

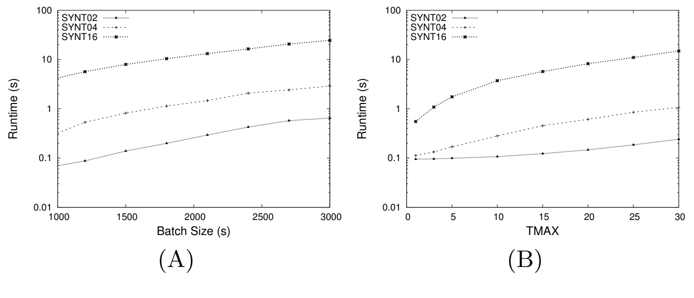


Fig. 6. Runtime w.r.t. T ($t_{max} = 10$) (A) and w.r.t. t_{max} ($T = 900$) (B).

them extend the Dynamic Time Warping (DTW) algorithm [6] that makes it possible to find an optimal time alignment between two time series. However, the time-series are warped non-linearly to be robust w.r.t. non-linear time variations. In our work, we consider linear transformations to find out dependencies as well as their most specific time-delay intervals. On another hand, temporal pattern mining [2] extracts frequent patterns among a set of sequences of time-point based events. To discover more sophisticated relations than the “before” / “after” one, interval-based events are considered to find complex relations using Allen’s algebra [10]. Incorporating statistical metric like χ^2 test within the pattern mining process is a well-studied issue [7]. But these measures are often considered in addition to others such as confidence and support measures. In this paper, this statistical assessment is used to automatically set the thresholds.

6 Conclusion

Our work identifies temporal dependencies between interval-based events. Our approach is robust to the temporal variability of events and characterizes the time intervals during which the events are dependent. As several intervals may redundantly describe the same dependency, the approach retrieves only the few most specific ones. The experiments show that the pruning techniques are very efficient and speed up the running time by a factor between 2 and 60.

References

1. C. C. Aggarwal, editor. *Data Streams - Models and Algorithms*, volume 31 of *Advances in Database Systems*. Springer, 2007.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pages 3–14, 1995.
3. M. Böttcher, F. Höppner, and M. Spiliopoulou. On exploiting the power of time in data mining. *SIGKDD Explorations*, 10(2):3–11, 2008.
4. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.
5. C. Jermaine. Finding the most interesting correlations in a database: how hard can it be? *Inf. Syst.*, 30(1):21–46, 2005.
6. E. J. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3):358–386, 2005.
7. S. Morishita and J. Sese. Traversing itemset lattice with statistical metric pruning. In *PODS*, pages 226–236, 2000.
8. K. Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random. *Psychology Magazine*, 1:157–175, 1900.
9. X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Min. Knowl. Discov.*, 26(2):275–309, 2013.
10. S.-Y. Wu and Y.-L. Chen. Mining nonambiguous temporal patterns for interval-based events. *IEEE Trans. Knowl. Data Eng.*, 19(6):742–758, 2007.

Using One-Versus-All classification ensembles to support modeling decisions in data stream mining

Patricia E.N. Lutu
Department of Computer Science,
University of Pretoria, South Africa
Patricia.Lutu@up.ac.za

Abstract. Data stream mining is the process of applying data mining methods to a data stream in real-time in order to create descriptive or predictive models. Due to the dynamic nature of data streams, new classes may emerge as a data stream evolves, and the concept being modelled may change with time. This gives rise to the need to continuously make revisions to the predictive model. Revising the predictive model requires that labelled training data should be available. Manual labelling of training data may not be able to cope with the speed at which data needs to be labelled. This paper proposes a predictive modeling framework which supports two of the common decisions that need to be made in stream mining. The framework uses a One-Versus-All (OVA) ensemble whose predictions make it possible to decide whether a newly arrived instance belongs to a new unseen class and whether an instances require manual labelling or not.

Key words: data mining, stream mining, OVA classification, ensemble classification, instance labelling.

1. Introduction

A data stream is defined as a continuous ordered sequence of data items. Data stream mining is defined as the process of applying data mining methods to a data stream in real-time in order to create descriptive or predictive models for the process that generates the data stream [1], [2], [3]. Due to the dynamic nature of data streams, new classes may emerge as a data stream evolves. Secondly, the concept being modelled may change abruptly or gradually with time [4], [5], [6]. The arrival of new classes and the changes in the concept being modelled give rise to the need to continuously make revisions to, or completely rebuild the predictive model when these changes are detected. Rebuilding the predictive model requires that labelled training data should be available. Data labelling for stream mining needs to be a fast process since stream data may arrive at a very high speed. Traditional methods of labelling training data may not be able to cope with the speed at which data needs to be labelled.

Given the foregoing discussion, various decisions need to be made for the predictive modeling process in stream mining [7]. Three of these decisions are: (1) detecting the emergence of new classes (2) determining concept change / drift and (3) deciding which newly arrived instances should be manually labelled for future model revisions. This paper proposes a predictive modeling framework for stream mining based on binary classification. The main objective of this framework is to reduce the number of instances that require manual labelling. Other secondary objectives are to detect the arrival of instances that belong to new classes, and to give an indication of the occurrence of concept change. The framework uses an ensemble model composed of One-Versus-All (OVA) base models [8], [9], [10]. Each base model specialises in the prediction of instances in one region of the instance (measurement) space. The predictions produced by the ensemble make it possible to decide whether a predicted class label should be confidently assumed to be correct, or whether it could be confusion between two classes so that manual verification is required, or whether the ensemble is not equipped to provide a class label for the instance. The experimental results demonstrate that the proposed framework has a high potential to reduce the number of instances that require manual labelling. The rest of this paper is organised as follows: Section 2 provides the background to the reported research. Section 3 presents the proposed stream mining framework. The experimental results are presented in Section 4. Section 5 concludes the paper.

2. Background

2.1 Challenges in stream mining

One major challenge for mining data streams is due to the fact that it is infeasible to store the data stream in its entirety. This problem makes it necessary to select and use training data that is not outdated for the mining task. The second challenge for stream mining is due to the phenomenon of concept drift, which is defined as the gradual or rapid changes in the concept that a mining algorithm attempts to model [1], [2], [3]. Given these challenges, there is a need to continuously revise the model. The third challenge is due to the fact that for predictive classification modelling there is a need to rapidly and continuously provide training data which consists of instances that are labelled with the classes.

One approach to selecting data for mining data streams is called the sliding window approach. A sliding window, which may be of fixed or variable width, provides a mechanism for limiting the data used for modeling to the most recent instances. The main advantage of this technique is to prevent stale data from influencing the models obtained in the mining process [5], [6]. Two main problems with this approach are that firstly, for predictive modeling, there is an in-built assumption that labelled training data is rapidly available. Secondly, the predictive model needs to be continuously recreated as the window slides. Data stream instances do not typically arrive in an independent and identically distributed (iid) fashion. It is possible for instances of one class to arrive over a prolonged period of time. When this is the case, it may become infeasible to employ the sliding window approach, as the model could end up being trained on instances of one class only! A second approach to stream mining is to employ ensemble classification. Ensemble models for stream mining resolve the problems created by the sliding window approach by creating a new base model only when a new batch of labelled instances arrives and keeping base models that are trained on both old and new instances. At any point in time, available training data is also very likely to be imbalanced in the class distributions due to the non-iid nature of a data stream. OVA ensemble classification has been proposed by Hashemi et. al [9] for resolving this problem.

Masud et. al [11], Zhu et. al [12] and Zhang et. al [13] have all observed that, for stream mining, manual labelling of data is a costly and time consuming exercise. In practice it is not possible to label all stream data, especially when the data arrives at a high speed. It is therefore common practice to label only a small fraction of the data for training and testing purposes. Masud et. al [11] have proposed the use of ensemble classification models based on semi-supervised clustering, so that only a small fraction (5%) of the data needs to be labelled for the clustering algorithm. Zhu et. al [12] have proposed an active learning framework for solving the instance labelling problem. The main challenge of active learning is to identify the most informative instances that should be labelled in order to achieve the highest accuracy.

2.2 Ensemble classification for stream mining

Several ensemble classification methods for stream mining have been reported in the literature. These methods include the use of All-Classes-At-once (ACA) base models (e.g. [14], [15]), the use of All-Versus-All (AVA) base models (e.g. [16]) or the use of OVA base models (e.g. [9]). The main objective of ACA ensembles for stream mining is to (1) avoid overfitting, (2) avoid the use of a very limited amount of training data as is the case for the sliding window model, and (3) reduce the computational effort of revising the whole model when concept change/drift is detected [14], [15]. Examples of ACA ensemble frameworks that have been reported in the literature are the streaming ensemble algorithm (SEA) [17], the accuracy-weighted ensemble (AWE) [14], and the dynamically weighted majority (DWM) ensemble [15]. The main objectives of ensembles which use binary classification base models for stream mining is to (1) provide an easy approach to handle imbalanced training data (2) provide a fast method of model revision and (3) increase predictive accuracy [9]. Hashemi et. al [9] have studied the use of OVA classification tree ensembles for stream mining and have concluded that these ensembles provide fast and accurate performance compared with state-of-the art stream classification algorithms e.g. CVFDT [18]. They have also observed that OVA ensembles provide very fast reaction to concept drift / change since only the base models for the class(es) for which concept drift has been detected need to be changed.

2.3 Active learning

It was stated above that manual labelling of instances for model training is a costly and time consuming exercise. Active learning is a branch of machine learning concerned with the automated selection of the most useful instances that should be manually labelled by a human expert [19]. Settles [19] has observed that the original motivation for active learning was to enable a learning algorithm to choose the data from which it learns, so that it would perform better with less training. More recently, a second motivation for active learning is to enable a learning algorithm to learn more economically. So, an active learning scheme automatically identifies the useful instances for manual labelling and then requests a human expert for the class labels. Active learning schemes are typically characterised by the methods they employ to select the instances for labelling. Two selection methods that are generally suited to stream mining and are not tied to any specific classification algorithm are uncertainty sampling and query by committee (QBC) [19].

In the context of active learning a query is defined as an unlabelled instance that is passed to a human expert for labelling. For the uncertainty sampling scheme, the learner queries the instances about which it is most uncertain on how to label. One approach to uncertainty sampling is to query those instances whose posterior probability of belonging to any of the classes is very close to the probability for random guessing which is 0.5 for a 2-class problem [19]. A second approach is to use margin sampling where instances selected for querying are those for which the posterior probabilities of the winning class and the second best class are very close (i.e. the difference is less than a user specified margin) [19]. A third approach uses entropy as a measure of the level of variability in the assignment of prior probabilities for the predicted classes. The lower the variability, the more uncertain the prediction. The query by committee (QBC) scheme maintains a committee of classifiers all trained on the same training data but representing competing hypotheses (target functions) [19]. Each committee member is then requested to vote on the class labels of the query instances. The most informative instances are those on which the committee members disagree the most. Measures of disagreement for QBC schemes include the entropy vote and the Kullback-Leibler divergence measure [19].

Settles [19] has observed that even though active learning provides a practical solution to the cost reduction for instance labelling, it suffers from two major weaknesses. The first major weakness is due to the fact that the training instances are a biased distribution and not an iid sample which represents the underlying natural density of the available data. The second major weakness is due to the computationally intensive algorithms for active learning. For each instance that is processed by the algorithm, a value must be computed for the measure of informativeness, measure of disagreement, or some other measure.

3. Proposed stream mining framework

As stated above, the main objective for the research reported in this paper was to study methods for reducing the amount of effort for labelling training data. For this to be made possible, the approach that was adopted was to create a stream mining model consisting of OVA base models where each base model can predict one class for a *k*-class prediction task. This section presents the motivation for using OVA classification as well as the details of for the proposed framework.

3.1 OVA classification

OVA classification [8] is a method of classification where a *k*-class prediction problem is decomposed into *k* sub-problems for classification. Base classifiers,

$OVA_1,...,OVA_k$, are created and combined into one ensemble where the base model predictions are combined and the best prediction is selected based on the value of a decision function. OVA classification was selected as the method to be studied for the proposed framework for the following reasons: Firstly, each OVA classifier solves a 2-class problem which greatly simplifies the base models and can provide very high levels of predictive accuracy compared to a single *k*-class model [10]. Several researchers (e.g. [20]) have conducted studies which indicate that the use of simple models has the potential to reduce the bias and variance components of the prediction error. In the context of the proposed framework, the reduction of the number of instances that need manual labelling requires the use a predictive model that provides very high levels of predictive accuracy. Secondly, OVA classification enables the creation of base models where each base model specialises in one region of the instance space. Given this capability, it is ideally easy to determine the region where a test or query instance lies based on the nature of the OVA base model predictions. If only one base model predicts a class then the instance comes from the decision region for the class that the base model is designed to predict. If two base models predict their classes then this can be interpreted to mean that the instance is located in the decision boundary for the two classes and is therefore difficult to predict. If none of the base models predict their classes this can be interpreted as an instance which is either noise or belongs to a new class that the ensemble is not designed to predict. Thirdly, an OVA ensemble simplifies the necessary model revisions for stream mining. It is not uncommon for instances of only one class or a subset of the classes to arrive for a prolonged period of time. It is also important to include the most recently labelled training instances to ensure that the model is always up to date as much as possible. OVA modeling supports these stream mining aspects since the base model for a class can be quickly revised whenever a sufficiently large amount of training data for the class becomes available.

3.2 Making stream mining decisions

Figure 1 depicts the components and outputs of the framework. The predictive model for the framework consists of OVA base models and a combination / decision algorithm. The combination algorithms receives the base model predictions of the form (*class, score*) for each test or query instance and produces an output which is a triple (*class, score, category*) for the instance. An instance is assigned to one of three categories: *sure*, *notsure*, or *dontknow*. When only one OVA_i base model provides a prediction for class C_i, and all the other OVA base models predict ‘other’, then the prediction is considered to be a confident prediction so that no manual labelling is needed. This decision is contingent on the base model for the class having a very high level of predictive accuracy. For this outcome, the output of the combination / decision algorithm is (C_i, *score*$_i$, *sure*). When several OVA base models predict the classes they are designed to predict, this is an indication that there is confusion in the ensemble so that the ensemble is not sure about which class the instance belongs to. In practice, the prediction with the highest score is taken as the ensemble prediction [10]. In the proposed framework, the output of the combination / decision algorithm for this outcome, is (C_i, *score*$_i$, *notsure*) where C_i is the predicted class with the highest score. When all the OVA base models predict 'other', it is assumed that the instance belongs to a new class. The output for this outcome is (*other, 0, dontknow*).

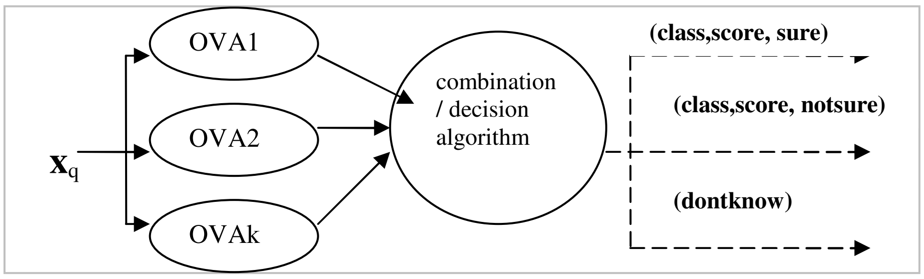


Figure 1: The components and outputs of the proposed framework

3.3 Problems that are solved by the proposed framework

It was stated in Section 2.3 that active learning results in the usage of training samples which have a biased probability distribution. This could be problematic for real-world applications for which classification models are only useful if the training data samples are iid so that they reflect the underlying probability density. This is for example the case if lift analysis of the classification results is required. The proposed framework makes it possible for training data to be obtained from all the regions of the instance space while at the same time reducing the number of instances that require manual labelling. Active learning involves intensive computations for the measure of informativeness, or measure of disagreement, or some other measure. These measures need to be computed for every instance. For large quantities of data such as typically found in stream mining, this computational overhead may not be feasible. The proposed framework does not require any intensive computations for deciding on which instances require manual labelling.

4. Experiments to study the ensemble performance

This section reports the results of the exploratory experiments that were conducted to study the performance of the proposed framework in terms of predictive accuracy and reliable labelling of data stream instances.

4.1 Dataset and classification algorithm

The See5 classification modeling software [21] was used for the exploratory studies using the forest cover type dataset available from the UCI KDD archive [22]. Zliobaite et. al [23] have observed that when the instances in the forest cover type dataset are sorted on the elevation attribute (feature) one simulates the effect of gradual concept change/drift as the elevation (altitude) increases, causing class descriptions to change, some classes to disappear, and new classes to emerge. The sorted dataset also exhibits other characteristics that provide challenges in data stream mining. One such challenge is the extreme imbalance of class distributions over different time periods. For these reason, the forest cover type dataset was used for the exploratory studies. The dataset consists of 581,012 instances, 54 features and 7 classes (C1, C2,..,C7). After sorting the dataset on the elevation feature, a new feature (called ID) was added to the dataset, with values in the range [1,581012] as a pseudo-timestamp. Figure 2 shows a plot of the class distributions for the pseudo-time periods used in the studies. Each time period represents the 'arrival' of 30,000 instances. It is clear from Figure 2 that classes C1 and C2 are the majority classes and that these classes are present in the data stream for the duration of the data arrival. Classes C3, C4, C5 and C6 initially appear in the stream and then disappear, while class C7 is not initially present in the stream but appears towards the end.

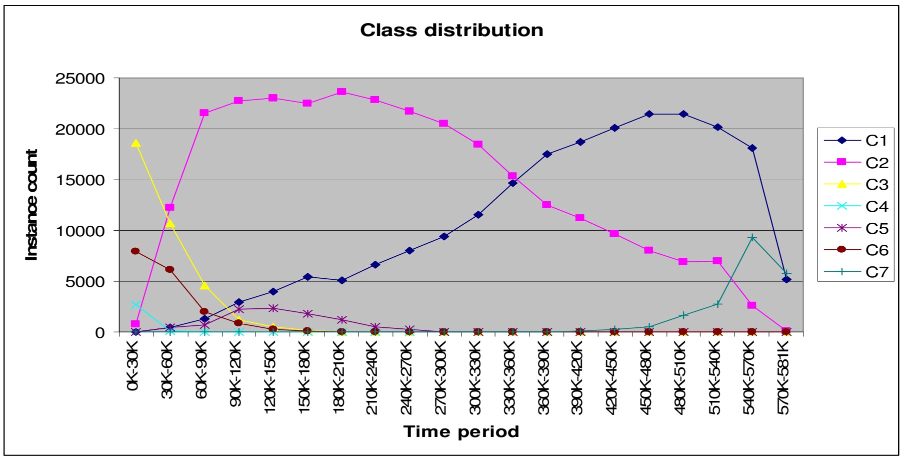


Figure 2: Class distribution for the Cover type data stream

4.2 Creation of the initial model

The top 150,000 instances of the sorted forest cover type dataset were used for the creation and testing of the base models for the initial ensemble. The top 120,000 instances were divided into four batches of training instances and were used to select the training data for each OVA base model. The next 30,000 instances were used to initially test the predictive performance of the base models and the ensemble. Table 1 shows the instance counts by class for the top 4 time periods. It is clear that not all classes are present for each time period. Due to the imbalanced class distributions, training data for the base models was taken from different time periods with the criterion of always using the most recent data for each class. The OVA1 training data was taken from period 30K-120K. The OVA2 training data was taken from period 90K-120K. For OVA3, training data was taken from period 30K-120K. For OVA4 training data was taken from period 0K-30K. For OVA5 training data was taken from period 30K-120K. For OVA6 training data was taken from period 60K-120K.

Table 1. Class distribution for the top 120,000 instances

	Class counts for training data for time period:				
class	0K-30K	30K-60K	60K-90K	90K-120K	Total for class
C1	0	403	1,283	2,927	4,613
C2	747	12,248	21,522	22,716	57,233
C3	18,580	10,683	4,579	1,235	35,077
C4	2,699	48	0	0	2747
C5	0	474	665	2,244	3,383
C6	7,974	6,144	1,951	878	16,947
C7	0	0	0	0	0
Total	30,000	30,000	30,000	30,000	120,000

4.3 Experimental results

Exploratory experiments were conducted to answer the following questions: (1) Does the proposed framework provide reliable categories of 'sure', 'notsure' and 'dontknow' categories of class predictions? (2) Does the proposed framework provide a practically significantly reduction in the number of data stream instances that require manual labelling? (3) Can the proposed framework provide reliable information on the emergence of new classes? (4) Can the proposed framework provide useful information that can lead to the detection of concept change and/or concept drift? The OVA ensemble design presented in the last section was tested on the next 461,012 instances of the data stream. For analysis purposes, the test instances were divided into sixteen time periods of 30,000 instances each. Additionally the OVA1 and OVA2 base models were re-trained and used at the beginning of time periods 300K-330K and 450K-480K. The OVA7 base model was added to the ensemble at the beginning of time period 540K-570K. Figure 2 shows the percentages of the prediction categories 'sure', 'notsure' and 'dontknow' for the OVA ensemble, for the sixteen time periods for all classes. The description of how these categories are determined was given in Section 3.2. The analysis results indicate that for all classes taken together the levels of 'sure' predictions are very high.

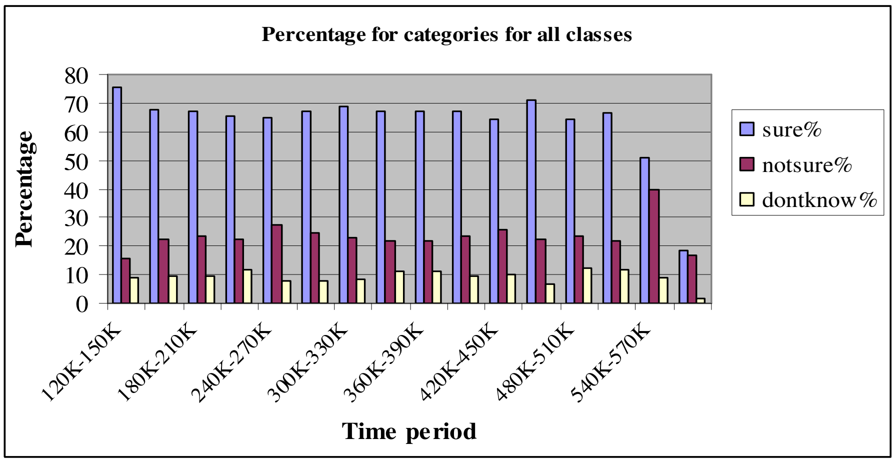


Figure 2: Classification categories for all classes for 16 time periods

The ‘sure’ predictions for the majority classes C1 and C2 were further analysed to establish the levels of reliability for each class. Figures 3 and 4 show the results of the analysis. The results of Figure 3 indicate that even though for each time period there is a high level of ‘sure’ predictions for class C1, the number of ‘sure’ predictions that are also correct predictions is initially very low but increases in the last five time periods. On the other hand, the results of Figure 4 indicate that for class C2 there is a high level of ‘sure’ predictions which are also correct predictions for all time periods.

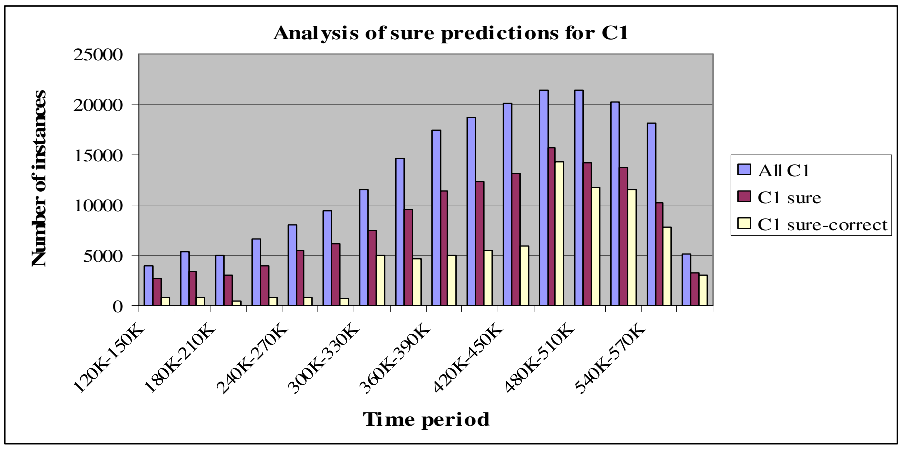


Figure 3: Analysis of sure predictions for class C1 for 16 time periods

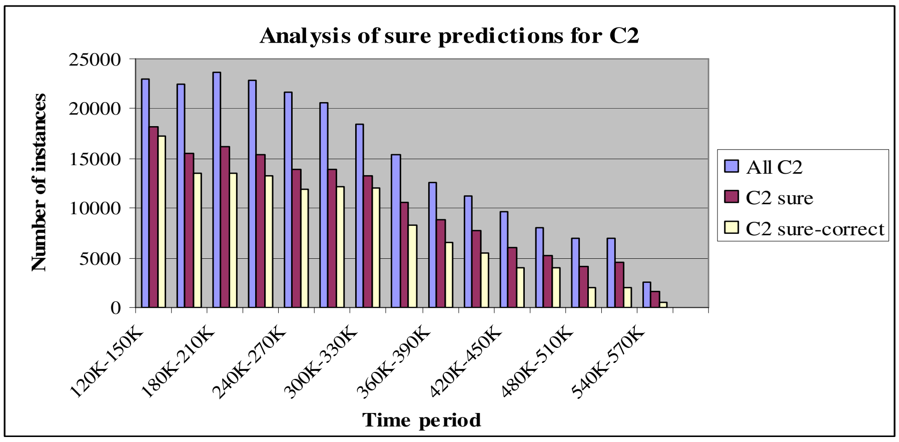


Figure 4: Analysis of sure predictions for class C2 for 16 time periods

4.5 Supporting stream mining modeling decisions

The results of the last section lead to the following conclusions: (1) If a class is easy to predict correctly (e.g. class C2) then the predictions for that class are largely reliable so that they can be treated as ‘sure’ predictions. For purposes of instance labelling for stream mining, it is not necessary to perform manual labelling of the instances of that class. (2) If a class is difficult to predict correctly (e.g. class C1 in the first 11 time periods), then the predictions for that class should not be treated as reliable. For purposes of instance labelling for stream mining, it is necessary to perform manual labelling of the instances of that class. It was stated previously that two of the objectives of the proposed framework are to reduce the amount of stream data that requires manual labelling and to provide an indication when new classes have arrived. The discussion of the last section has indicated that query instances that are categorised as 'sure' predictions by the ensemble should be taken as correctly labelled, but only for those classes for which it has been established that the class is easy to predict correctly. Since the ease of prediction may changes with time, periodic testing is needed to establish the classes that are easy to predict at a given time. The testing should be done whenever any base model is revised. All instances that do not fall in the 'sure' category as predicted by an OVA base model for an easy class require manual labelling. Table 2 provides an analysis of the reduction in the required instance labelling for the sixteen time periods.

Table 2: Analysis of the reduction in the required instance labelling

Count for category for all classes for 16 time periods (initial model)							
Time period	sure-correct	sure-incorrect	notsure	dontknow	Needs labelling		% needs labelling
					no	yes	
120K-150K	18731	3897	4683	2689	18731	11269	37.6
150K-180K	14512	5838	6788	2862	14512	15488	51.6
180K-210K	14069	5987	7052	2892	14069	15931	53.1
210K-240K	14132	5545	6797	3526	14132	15868	52.9
240K-270K	12793	6746	8168	2293	12793	17207	57.4
270K-300K	12918	7220	7455	2407	12918	17082	56.9
300K-330K	16955	3712	6845	2488	16955	13045	43.5
330K-360K	12976	7152	6588	3284	12976	17024	56.7
360K-390K	11515	8682	6464	3339	11515	18485	61.6
390K-420K	10943	9177	7048	2832	10943	19057	63.5
420K-450K	9936	9325	7730	3009	9936	20064	66.9
450K-480K	18305	2974	6676	2045	18305	11695	39.0
480K-510K	13772	5586	7028	3614	13772	16228	54.1
510K-540K	13528	6457	6536	3479	13528	16472	54.9
540K-570K	8725	6598	11908	2769	8725	21275	70.9
570K-581K	3119	2420	4995	478	3119	7893	71.7

For the first time period, only 37.6% of data stream instances need manual labelling. As time progresses, the percentage of instances that need manual labelling gradually increases until model revision when the percentage drops again. This is evident for periods 300K-33K and 450K-480K where revised OVA1 and OVA2 base models were added to the ensemble. When a query (or test) instance is categorised as ‘dontknow’ this may be due to the fact that it belongs to an existing class but cannot be predicted correctly, or it may be because it belongs to a previously unseen (new) class. It was observed above that as time progresses, the number of instances that require manual labelling increases. This is an indication of the fact that the current ensemble is gradually finding it more and more difficult to confidently and correctly classify the data stream instances. This can be used to determine when base models revision is necessary. It was stated in Section 3.3 that the proposed framework makes it possible to economically obtain iid training data samples. Based on the discussion in this section, data for training can be obtained from the set of instances that are categorised as ‘sure’ predictions as well as the instances that are manually labelled.

5. Conclusions

The first question for the exploratory studies was: *Does the proposed framework provide reliable categories of ‘sure’, ‘notsure’ and ‘dontknow’ categories of class predictions?* The answer to this question is yes. The experimental results presented in Section 4 have demonstrated that the framework can indeed provide reliable categories that can be used determine whether manual instance labelling is required. The second question was: *Does the proposed framework provide a practically significant reduction in the number of data stream instances that require manual labelling?* Again, the answer is yes. The results of Table 2 have demonstrated that this is the case for the forest cover type data set. The third question was: *Can the proposed framework provide reliable information on the emergence of new classes?* Based on the data that was used for the experiments, there is no conclusive evidence to support a ‘yes’ answer to this question. It was observed that instances that are categorised as ‘dontknow’ may belong to a new class (class C7) or they may be mis-classifications. Further studies are needed before any conclusions can be made. The fourth question was: *Can the proposed framework provide useful information that can lead to the detection of concept change and/or concept drift?* The answer is a tentative ‘yes’. Gradual or sudden increases in the ‘notsure’ and ‘dontknow’ predictions provide an indication that there are changes in the concept for which the model was created. Again, further studies are needed before a firm answer can be given for this question.

References

1. Aggarwal, C.C.: Data Streams: Models and Algorithms, Kluwer Academic Publishers, Boston (2007).
2. Gao, J., Fan, W. and Han, J.: On appropriate assumptions to mine data streams: analysis and practice, Proceedings of the Seventh IEEE International Conference on Data Mining (ICDM 2007), IEEE Computer Society (2007).
3. Masud, M.M., Chen, Q. and Gao, J.: Classification and novel class detection of data streams in a dynamic feature space, Proceedings of European Conference on Machine Learning and Practices in Knowledge Discovery from Databases (ECML/PKDD 2010), LNAI, 337-352, Springer-Verlag (2010).

4. Gao, J., Fan, W., Han, J. and Yu, P.S.: A general framework for mining concept-drifting data streams with skewed distributions, Proceedings of the SDM Conference (2007).
5. Hebrial, G.: Data stream management and mining, In F. Fogelman-Soulié et al. (eds), Mining Massive Data Sets for Security, IOS Press (2008).
6. Gaber, M.M., Zaslavsky, A. and Krishnaswamy, S.: Mining data streams: a review, SIGMOD Record, vol. 34, no. 2, pp. 18- 26 (2005).
7. Bifet, A., Holmes, G., Kirkby, R, and Pfahringer, B.: Data stream mining: a practical approach, (2011).
8. Rifkin, R., Klautau, A.: In defense of one-vs-all classification. The Journal of Machine Learning Research 5 101-141 (2004).
9. Hashemi, S., Yang, Y., Mirzamomen, Z., and Kangavari, M.: Adapted one-versus-all decision trees for data stream classification, IEEE Transactions on Knowledge and Data Engineering, 21(5) (2009).
10. Lutu, P. E. N. and Engelbrecht, A. P.: Using OVA modeling to improve classification performance for large datasets. Expert Systems With Applications, 39 (4) 4358-4376 (2012).
11. Masud, M.M., Gao, J., Khan, L. and Han, J.: A practical approach to classifying data streams: training with limited amount of data. Proceedings of the 8th IEEE International Conference on Data Mining, pp. 929-934 (2008).
12. Zhu, X., Zhang, P., Lin, X. and Shi, Y.: Active learning from data streams. Proceedings of the 7th IEEE International Conference on Data Mining, pp. 757-762 (2007).
13. Zhang, P., Zhu, X.and Guo, L., Mining data streams with labelled and unlabelled training examples. Proceedings of the 9th IEEE International Conference on Data Mining, pp. 627-636, 2009.
14. Wang, H., Fan, W., Yu, P.S. and Han, J.: Mining coNcept-drifting data streams using ensemble classifiers. Proceedings of the ACM Special Interests Group on Knowledge Discovery in databases,SIGKDD'2003, pp.226-235, Washington DC (2003).
15. Kolter, J.Z. and Maloof, M.A.: Dynamic weighted majority: an ensemble method for drifting concepts, Journal of Machine Learning Research, 8, 2755-2790 (2007).
16. Gama, J., Medas, P. and Rocha, R.: Forest trees for on-line data, Proceedings of the ACM Symposium on Applied Computing, pp. 632-636, Cyprus (2004).
17. Street, W.N. and Kim, Y.: A streaming ensemble algorithm (SEA) for large-scale classification. Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 377-382. ACM Press, New York (2001).
18. Hulten, G., Spencer, L., and Domingos, P.: Mining time-changing data streams. Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 97-106, San Francisco, CA (2001).
19. Settles, B.: Active learning literature survey, Department of Computer Sciences Technical Report 1648, University of Wisconsin-Madison (2009). Available at: http://active-learning.net/
20. Dietterich, T. and Kong, E.: Machine learning bias, statistical bias, and statistical variance of decision tree algorithms Technical Report. Corvallis, Oregon, Department of Computer Science, Oregon State University (1995).
21. Quinlan, J. R.: An Informal Tutorial, Rulequest Research. URL: http://www.rulequest.com (2004). (accessed: 28 October, 2005).
22. Bay, S.D., Kibler, D., Pazzani, M.J. and Smyth, P.: "The UCI KDD archive of large data sets for data mining research and experimentation, ACM SIGKDD, vol. 2, no. 2, pp. 81-85 (2000).
23. Zliobaite, I., Bifet, A., Pfahringer, B. and Holmes, G.: Active learning with drifting streaming data. IEEE Transactions on Neural Networks and Learning Systems, 25(1), pp. 27-39, 2014 (2011).

Mining Positional Data Streams

Jens Haase and Ulf Brefeld

Knowledge Mining & Assessment Group
Technical University of Darmstadt, Germany

Abstract. We study frequent pattern mining from positional data streams. Existing approaches require discretised data to identify atomic events and are not applicable in our continuous setting. We propose an efficient trajectory-based preprocessing to identify similar movements and a distributed pattern mining algorithm to identify frequent trajectories. We empirically evaluate all parts of the processing pipeline.

1 Introduction

Recent advances in telecommunication, sensing, and recording technologies allow for storing positions from moving objects at large scales in (near) real time. Analysing positional data streams is highly important in many applications; examples range from navigation and routing systems, network traffic, animal migration/tracking to tactics in team sports.

In this paper, we focus on identifying frequent movement patterns in positional data streams that consist of a possible infinite sequence of coordinates. Existing approaches to frequent pattern mining [3, 17] use identities of atomic events to define sequences (episodes) [12]. In positional data, events correspond to sequences of positions (i.e., trajectories) and due to the continuous domain it is very unlikely to observe a trajectory twice. Instead, we observe a multitude of different trajectories that give rise to an exponentially growing set of possibly frequent sequences. Consequentially, mining positional data can only be addressed in the context of big data.

Our contribution is threefold: (i) To remedy the absence of matching atomic events, we propose an efficient preprocessing of the positional data using locality sensitive hashing and approximate dynamic time warping. (ii) To process the resulting near-neighbour trajectories we present a frequent pattern mining algorithm that generalises Achar et al. [1] to positional data. (iii) We present a distributed algorithm for processing positional data at large-scales. Empirically, we evaluate all stages of our approach on positional data of a real soccer game where cameras and sensors realise a bird's eye view of the pitch that allows for locating the players and the ball several times per second.

2 Related Work

Spatio-temporal data mining aims to extract the behaviour and relation of moving objects from (positional) data streams and is frequently used in computational biology for mining animal movements. Trajectory-based patterns are first introduced by [5]. These patterns represent a set of individual trajectories that share the property of visiting the same sequence of places within similar travel time. Trajectory-based approaches use a discretisation of the movements to identify places that are also known beforehand. Our contribution considers a continuous generalisation: every coordinate on the pitch is a place of interest and trajectories are relations between coordinates and travel time.

Event sequence mining has been introduced by [3] as a problem of mining frequent sequential patterns in a set of sequences. Sequential pattern mining discovers frequent subsequences as patterns in a sequence database. The most common example is the cart analysis proposed by [3]. Frequent episode discovery is a technique to describe and find patterns in a stream of events [12]. Achar et al. [1] propose the first approach to mine unrestricted episodes. Our approach generalises [1] to mining positional data streams.

The behaviour of individual players is analysed by [6] and [7]. [6] analyse groups of players and their behaviour using self organising maps on top of the positional data. Every neuron of the network represents a certain area of the pitch. Thus, whenever a player moves into such an area, the respective neuron is activated. Similarly, [7] uses positional data to assess player positions in particular areas of the pitch, such as catchable, safe or competing zones. Prior work for instance also utilises positional data to identify tactical patterns [13]. However, these approaches usually focus on detecting *a priori* known patterns in the data stream. By contrast, we leverage the findings of *trajectory pattern* and *frequent episode discovery* to devise a purely data-driven approach to find tactical patterns in positional data without making any assumptions on zones, tasks or movements.

3 Efficiently Finding Similar Movements

3.1 Representation

Given a positional data stream $\mathcal{D}$ with ℓ objects $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_\ell$. Every object $\boldsymbol{o}_i$ is represented by a sequence of coordinates $\mathcal{P}_i = \langle \boldsymbol{x}_1^i, \boldsymbol{x}_2^i, \ldots \rangle$ where $\boldsymbol{x}_t = (x_1, x_2, \ldots, x_d)^\top$ denotes the position of the object in d-dimensional space at time t. A trajectory or movement of the i-th object is a subset $\boldsymbol{p}_{[t,t+m]} \subseteq \mathcal{P}_i$ of the stream, e.g., $\boldsymbol{p}_{[t,t+m]} = \langle \boldsymbol{x}_t^i, \boldsymbol{x}_{t+1}^i, \ldots, \boldsymbol{x}_{t+m}^i \rangle$, where m is the length of the trajectory. In the remainder, the time index t is omitted and each element of a trajectory is indexed by offsets $1, \ldots, m$.

For generality, we focus on finding similar trajectories where (i) the exact location of a trajectory does not matter (*translation invariance*), (ii) the range of the trajectory is negligible (*scale invariance*), and where turns such as left or right are considered identical (*rotation invariance*). Note that, depending on the application at hand, one or more of these requirements may be inappropriate and can be dropped by altering the representation accordingly.

Using the requirements (i)-(iii) gives rise to the so-called angle/arc-length representation [16] of trajectories that represents movements as a list of tuples of angles θ_t and distances $\boldsymbol{v}_t = \boldsymbol{x}_t - \boldsymbol{x}_{t-1}$. The difference $\boldsymbol{v}_t$ is called the *movement vector* at time t and the angles are computed with respect to a (randomly drawn) reference vector $\boldsymbol{v}_{ref} = (1, 0)^\top$. Transformed trajectories are normalised by subtracting the average so that $\theta_i \in [-\pi, +\pi]$ for all i and by normalising the total distance to one. Finally, we discard the difference vectors and represent trajectories solely by their sequences of angles, $\boldsymbol{p} \mapsto \tilde{\boldsymbol{p}} = \langle \theta_1, \ldots, \theta_n \rangle$.

3.2 Approximate Dynamic Time Warping

Recall that pairs of trajectories may contain phase shifts, that is, a movement may begin slowly and then speeds-up while another starts fast and then slows down towards the end. Such phase shifts are well captured by alignment-based similarity measures such as dynamic time warping [14].

Dynamic time warping (DTW) is a non-metric distance function that measures the distance between two sequences and is often used in speech recognition problems. Given two sequences $\boldsymbol{s} = \langle s_1, \ldots, s_n \rangle$ and $\boldsymbol{q} = \langle q_1, \ldots, q_m \rangle$ and a cost function $cost(s_i, q_j)$ detailing the costs of matching s_i with q_j. The goal of dynamic time warping is to find an alignment of sequences $\boldsymbol{s}$ and $\boldsymbol{q}$ that has minimal costs subject to *boundary*, *continuity*, and *monotonicity* constraints [9]. Note that che cost function *cost* can be arbitrarily defined and the complexity of DTW is $\mathcal{O}(|\boldsymbol{s}||\boldsymbol{q}|)$ which is prohibitive for mining positional data streams.

Efficient approximations of dynamic time warping can be obtained by lower bounds. The rationale is that lower bound functions can be computed in less time and are therefore often used as pruning techniques in applications like indexing or information retrieval. The exact DTW computation only needs to be carried out if the lower bound value is above a given threshold. We make use of two lower bound functions, f_{kim} [10] and f_{keogh} [8], that are defined as follows: f_{kim} focuses on the first, last, greatest and smallest values of two sequences [10] and can be computed in $\mathcal{O}(m)$:

$$f_{kim}(\boldsymbol{s}, \boldsymbol{q}) = \max \left\{ |s_1 - q_1|, |s_m - q_m|, |\max(\boldsymbol{s}) - \max(\boldsymbol{q})|, |\min(\boldsymbol{s}) - \min(\boldsymbol{q})| \right\}.$$

If the greatest and the smallest entries are normalised to a specific value their computation can be ignored and the time complexity reduces to $\mathcal{O}(1)$. The second lower bound f_{keogh} [8] uses minimum $\ell_i = \min(q_{i-r}, \ldots, q_{i+r})$ and maximum values $u_i = \max(q_{i-r}, \ldots, q_{i+r})$ for sub-sequences of the query $\boldsymbol{q}$ where r is a user defined threshold. Trivially, $u_i \geq q_i \geq \ell_i$ holds for all i and the lower bound f_{keogh} is given by $f_{keogh}(\boldsymbol{q}, \boldsymbol{s}) = \sqrt{\sum_{i=1}^{m} c_i}$ where $c_i = (s_i - u_i)^2$ if $s_i > u_i$, $c_i = (s_i - \ell_i)^2$ if $s_i < \ell_i$, and $c_i = 0$ otherwise. The function f_{keogh} can also be computed in $\mathcal{O}(m)$. The result is a non-metric distance function that only violates the triangle inequality of a metric distance.

3.3 An N-Best Algorithm

Given a trajectory $\boldsymbol{q} \in \mathcal{D}$, the goal is to find the most similar trajectories in $\mathcal{D}$. Trivially, a straight forward approach is to compute the DTW values of $\boldsymbol{q}$ for all trajectories in $\mathcal{D}$ and sort the outcomes accordingly. However, this requires $|\mathcal{D}|$ DTW computations, each of which is quadratic in the length of the trajectories, and renders the approach clearly infeasible.

We now sketch how to compute the N most similar trajectories for a given query $\boldsymbol{q}$ efficiently by making use of the lower bound functions f_{kim} and f_{keogh}. The algorithm begins with computing the DTW distances of the first N entries in the database and stores the entry with the highest distance to $\boldsymbol{q}$. A loop over the remaining trajectories in $\mathcal{D}$ first applying the lower bound functions f_{kim} and f_{keogh} to efficiently filter irrelevant movements before using the exact DTW distance for the remaining candidates. Every trajectory, realising a smaller DTW distance than the current maximum, replaces its peer; auxiliary variables *maxdist* and *maxind* are updated accordingly. Note that the complexity of the algorithm is linear in the number of trajectories in $\mathcal{D}$. In the worst case, the sequences are sorted in descending order by the DTW distance, which requires to compute all DTW distances. In practice much lower run-times are observed.

A crucial factor is the tightness of the lower bound functions. The better the approximation of the DTW, the better the pruning. For $N = 1$, the maximum value drops faster towards the lowest possible value. By contrast, setting $N = |\mathcal{D}|$ requires to compute the exact DTW distances for all entries in the database. Hence, in most cases, $N \ll |\mathcal{D}|$ is required to reduce the overall computation time. The computation can trivially be distributed with Hadoop; computing distances is performed in the mapper and sorting is done in the reducer.

3.4 Distance-based Hashing

An alternative to the introduced N-Best algorithm provides locality sensitive hashing (LSH). A general class of LSH functions are called distance-based hashing (DBH) that can be used together with arbitrary spaces and (possibly non-metric) distances [4]. The hash family is constructed as follows. Let $h : \mathcal{X} \to \mathbb{R}$ be a function that maps elements $x \in \mathcal{X}$ to a real number. Choosing two randomly drawn members $x_1, x_2 \in \mathcal{X}$, the function h is defined as

$$h_{x_1,x_2}(x) = \frac{dist(x, x_1)^2 + dist(x_1, x_2)^2 - dist(x, x_2)^2}{2\, dist(x_1, x_2)}.$$

The binary hash value for x simply verifies whether $h(x)$ lies in an interval $[t_1, t_2]$, that is $h^{[t_1,t_2]}_{x_1,x_2}(x) = 1$ if $h_{x_1,x_2}(x) \in [t_1, t_2]$ and $h^{[t_1,t_2]}_{x_1,x_2}(x) = 0$ otherwise. where the boundaries t_1 and t_2 are chosen so that the probability that a randomly drawn $x \in \mathcal{X}$ lies with 50% chance within and with 50% chance outside of the interval. Given the set $\mathcal{T}$ of admissible intervals and hash function h, the DBH family is defined as the set of all admissible hash functions $h^{[t_1,t_2]}$. Using random draws from $\mathcal{H}_{DBH}$, new hash families can be constructed using *AND*- and *OR*-concatenation.

We use DBH to further improve the efficiency of an N-Best algorithm by removing a great deal of trajectories before processing them. Given a query trajectory $\boldsymbol{q} \in \mathcal{D}$, the retrieval process first identifies candidate objects that

Algorithm 1 FSATransition(α, fsa, t, $events$)

```
 1: if $fsa.currentState.Open = \varnothing$ then
 2:   return $fsa$ {FSA is in final state}
 3: end if
 4: for $n \in sourceNodes(fsa.currentState.Open)$ do
 5:   for $e \in events$ do
 6:     if $e \sim nodeMapping_\alpha(n)$ then
 7:       $fsa.currentState.Open = fsa.currentState.Open \setminus n$
 8:       $fsa.currentState.Done = fsa.currentState.Done \cup n$
 9:       $fsa.lastTransition = t$
10:       if $fsa.startTime == undefined$ then
11:         $fsa.startTime = t$
12:       end if
13:       break inner loop {Only one possible similarity (injective episode)}
14:     end if
15:   end for
16: end for
17: return $fsa$
```

are hashed to the same bucket for at least one of the hash functions, and then computes the exact distances of the remaining candidates using the N-Best algorithm. As distance measure of the DBH hash family we use the lower bound f_{kim}. The computation is again easily distributed with Hadoop.

4 Frequent Episode Mining for Positional Data

The main difference between frequent episode mining and mining frequent trajectories from positional data streams is the definition of events. For positional data, every trajectory in the stream is considered an event. Thus, events may overlap and are very unlikely to occur more than just once. We resort to the previously defined approximate distance functions in the mining step.

An event stream is a time-ordered stream of trajectories. Every event is represented by a tuple (A, t) where A is an event and t denotes its timestamp. An *episode* α is a directed acyclic graph, described by a triplet $(\mathcal{V}, \leq, m)$ where $\mathcal{V}$ is a set of nodes, $\leq$ is a partial order on $\mathcal{V}$ (directed edges between the nodes), and $m : \mathcal{V} \to E$ is a bijective function that maps nodes to events in the event stream. We focus on *transitive closed episodes* [15] in the remainder, that is if node A is ordered before B $(A < B)$ there must be a direct edge between A and B, that is, $\forall A, B \in \mathcal{V}$ if $A < B \implies edge(A, B)$. The partial ordering of nodes upper bounds the number of possible directed acyclic graphs on the event stream. The ordering makes it impossible to include two identical (or similar) events in the same episode. Episodes that do not allow duplicate events are called *injective episodes* [1].

An episode α is called *frequent*, if it occurs often enough in the event stream. The process of counting the episode α consists of finding all episodes that are

Algorithm 2 Map(id, α)

```
1: $eventStream = loadEventStreamFormFile()$
2: $frequency = 0; fsas = \{new\ FSA\}$
3: for all $(t, events) \in eventsStream$ do
4:    for all $fsa \in fsas$ do
5:       $inStartState = inStartState(fsa)$
6:       $hasChanged = FSATransition(\alpha, fsa, t, events)$
7:       if $inStartState$ and $hasChanged$ then
8:          $fsas = fsas \cup new\ FSA$
9:       end if
10:      if $inFinalState(fsa)$ then
11:         $fsas = \{new\ FSA\}$
12:         $frequency+ = 1$
13:      else
14:         $fsas = RemoveAllOlderFSAsInSameState(fsas)$
15:      end if
16:   end for
17: end for
18: if $frequency >= userDefinedThreashold$ then
19:    $EMIT(blockstart - id(\alpha), \alpha)$
20: end if
```

similar to α. A sub-episode β of an episode α can be created by removing exactly one node n and all its edges from and to n; e.g., for the episode $A \rightarrow B \rightarrow C$ the sub-episodes are $A \rightarrow B$, $A \rightarrow C$ and $B \rightarrow C$. The sub-episode of a singleton is denoted by the empty set $\varnothing$.

Frequent episodes can be found by Apriori-like algorithms [2]. The principles of dynamic programming are exploited to combine already frequent episodes to larger ones [12, 11]. We differentiate between alternating *episode generation* and *counting* phases. Every newly generated episodes must be *unique*, *transitive closed*, and *injective*. Candidates possessing infrequent sub-episodes are discarded due to the downward closure lemma [1]. We now present novel counting and episode generation algorithms for processing positional data with Hadoop.

4.1 Counting phase

The *frequency* of an episode is defined as the maximum number of non-overlapping occurrences of the episode in the event stream [11].[1] Non-overlapping episodes can be detected and counted with finite state automata (FSAs), where every FSA is tailored to accept only a particular episode. The idea is as follows. For every episode that needs to be counted, an FSA is created and the event stream is processed by each FSA. If an FSA moves out of the initial state, a new FSA is created for possibly later occurring episodes and once the final state has been

[1] Two occurrences of an episode are said to be *non-overlapping*, if no event associated with one appears in between the events associated with the other.

Algorithm 3 Align(α, β)

```
Require: $|nodes(\alpha)| = |nodes(\beta)|$
 1: $f$ = int array of length $|nodes(\alpha)|$
 2: $used$ = boolean array of length $|nodes(\alpha)|$
 3: $n = 0$
 4: for $i = 1$ to $|nodes(\alpha)|$ do
 5:     $event_{i,\alpha} = m(\alpha)[i]$
 6:     $found$ = false
 7:     for $j = 1$ to $|nodes(\beta)|$ do
 8:         $event_{j,\beta} = m(\beta)[j]$
 9:         if (not $used[j]$) and $event_{i,\alpha} \sim event_{j,\beta}$ then
10:             $f[i] = j$
11:             $used[j]$ = true
12:             $found$ = true
13:         end if
14:     end for
15:     if $found$ = false then
16:         $f[i] = -1$
17:         $increment(n)$
18:     end if
19: end for
20: return $f, n$
```

reached, the episode counter is incremented and all FSA-instances of the episode are deleted except for the one still remaining in the initial state.

Algorithm 1 shows the FSA transition function that counts an instance of an episode. Whenever the FSA reaches its final state its frequency is incremented. As input, Algorithm 1 gets the fsa instance which contains the current state, the last transition time and the first transition time. Additionally, the appropriate episode, the current time stamp and the events starting at that time stamp are passed to the function. First, in case the FSA is already in the final state, the function returns without doing anything (line 1). Algorithm 1 iterates over all source nodes in the current state and all events that had happened at time t (line 4-5). Whenever there is an event e that is similar to the appropriate event of source node n (line 6), the FSA is traversed to the next state. The algorithm also keeps track of the start time and the last transition time to check the expiry time (line 9 and line 11).

The FSA transition function allows the definition of the counting algorithm shown in Algorithm 2 as a *map*-function for the Hadoop/MapReduce framework. The function first loads the event stream[2] (line 1) and initialises an empty FSA for every episode. Next, the event stream and the FSAs are traversed and passed to the FSA transition function. Whenever an FSA leaves the start state a new FSA must be added to the set of FSAs. This ensures that there is exactly one

[2] In practice one would read the event stream block wise instead of loading the whole data at once into memory. We chose the latter for ease of presentation.

Algorithm 4 Combine(α,β)

```
1: $\pi, n = Align(\alpha, \beta)$
2: if $n \neq 1$ then
3:     return $-1$
4: end if
5: $sum_\alpha = 0$; $sum_\beta = 0$
6: $sum = \frac{|\pi| \times (|\pi|-1)}{2}$
7: for $i = 1$ to $|\pi|$ do
8:     if $\pi[i] \geq -1$ then
9:         $sum_\alpha = sum_\alpha + i$
10:        $sum_\beta = sum_\beta + \pi[i]$
11:    end if
12: end for
13: return $(sum - sum_\alpha, sum - sum_\beta)$
```

FSA in a start state. In case an FSA reaches its final state, all other FSAs can be removed and the process starts again with only one FSA in start state. In case more than one FSA reaches the final state, Algorithm 2 removes all but the youngest one in final state as this one has higher chances to meet the expiry time constraints. The test for expiry time is not shown in the pseudo code. Instances violating the expiry time do not contribute to the frequency count. Neither do FSAs that associate overlapping events with the same object. Note that the general idea of the counting algorithm is very similar to [1]. However, due to the different notions of an event, many optimisation do not apply in our case.

Following [1] we also employ bidirectional evidence as frequencies alone are necessary but not sufficient for detecting frequent episodes. The entropy-based bidirectional evidence can be integrated in the counting algorithm, see [1] for details. We omit the presentation here for lack of space.

4.2 Generation phase

Algorithm 4 is designed to efficiently find the indices of the differentiated nodes of two episodes α and β. Therefore, it first tries to find the bijective mapping π, that maps each node (and its corresponding event) of episode α to episode β (line 1). In case such a complete mapping can not be found, π returns only the possible mappings and n contains the number of missing nodes in the mapping (see Algorithm 3). Episodes α and β are combinable, if and only if $n = 1$. The remainder of the algorithm finds the missing node indices by accumulating over the existing indices and by subtracting the accumulated result from the sum of all indices. This little trick finds the missing indices in time $O(n)$. The function returns the node indices that differentiate between α and β.

To prevent the computation of Algorithm 4 on all pairs of episodes, each episode is associated with its *block start identifier* [1]. The idea is the following. All generated episodes from an episode α share the same sub-episode. This sub-

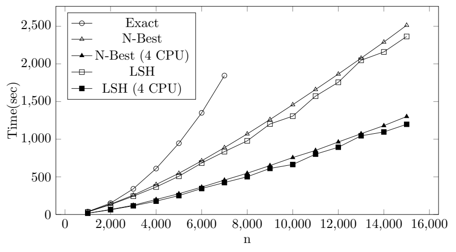
Algorithm 5 Reduce($blockstartId$, xs)

```
1: $k = -1$; $result = \varnothing$
2: for $i = 0$ to $|xs|$ do
3:     $\alpha = xs(i)$; $currentBlockStart = k + 1$
4:     for $j = i + 1$ to $|xs|$ do
5:         $\beta = xs(j)$
6:         if $\alpha.blockStart == \beta.blockStart$ then
7:             $candidates = Combine(\alpha, \beta)$
8:             for $c \in candidates$ do
9:                 if $transitiveClose(c)$ then
10:                    $c.blockStart = currentBlockStart$
11:                    $result = result \cup c$
12:                    $k = k + 1$
13:                end if
14:            end for
15:        else
16:            $break$
17:        end if
18:    end for
19: end for
20: $EMIT(id, result)$
```

episode is trivially identical to α as it originates from adding a node to α. The generation step thus takes only those episodes into account that possess the same block start identifier.

Given two combinable episodes α and β and the differentiated nodes a and b (found by Algorithm 4), it is now possible to combine these episodes to up to three new candidates, as described by [1]. The first candidate originates from adding node b to episode α including all its edges from and to b. The second candidate is generated from the first candidate by adding an edge from node a to node b and the third one adds an edge from b to a to the first candidate. In contrast to [1], we do not test wether all sub-episodes of each candidate are frequent as this would require an efficient lookup of all episodes which can be quite complex for positional data. Candidates with infrequent sub-episodes are allowed at this stage of the algorithm as they will be eliminated in the next counting step anyway.

The complete episode generation algorithm is shown in Algorithm 5. As input, a list of frequent episodes ordered by their block start identifier is given. The result of the algorithm is a list of new episodes that are passed on to the counting algorithm. In line 2 and line 4, all episode pairs are processed as long as they share the same block start identifier (line 6). Then, three possible candidates are generated (line 7) and kept in case they are transitive closed (line 9). Before adding it to the result set, the block start identifier of the new episode is updated (line 10). Analogously to the counting phase, domain specific constraints may be added to filter out unwanted episodes (e.g. in terms of expiry time, overlapping trajectories of the same object, etc.).



n	f_{kim}	f_{keogh}	LSH	Σ
1000	0%	0%	11.42%	11.42%
5000	0.28%	34.00%	16.33%	50.61%
10000	9.79%	41.51%	17.80%	60.10%
15000	17.50%	46.25%	11.82%	75.57%

Fig. 1. Left: Run-time. Right: Pruning efficiency.

5 Empirical Evaluation

5.1 Positional Data

We use positional data from the DEBS Grand Challenge 2013[3]. that is recorded from a soccer game with 8 players per team. We average player data over 100ms to obtain a single measurement for every player at each point in time. The set of trajectories is created by introducing sliding windows that begin every 500ms and last for one second. This procedure gives us 111.041 trajectories in total, 50.212 for team A, 50.245 for team B, and 10.584 for the ball.

5.2 Near Neighbour Search

The first set of experiments evaluates the run-time of the three distance functions *Exact*, *N-Best*, and *LSH*. Since the exact variant needs quadratically many comparisons in the length of the stream, we focus on only a subset of 15,000 consecutive positions of team A in the experiment and fix $N = 1000$. Figure 1 (left) shows the run-times in seconds for varying sample sizes.

Unsurprisingly, the computation time of the exact distances grows exponentially in the size of the data. By contrast, the N-Best algorithm performs slightly super-linear and significantly outperforms its exact counterpart. Pre-filtering trajectories using LSH results in only a small additional speed-up. The figure also shows that distributing the computation significantly improves the run-time of the algorithms and indicates that parallelisation allows for computing near-neighbours on large data sets very efficiently. The observed improvements in run-time are the result of a highly efficient pruning strategy (Figure 1, right).

Figure 2 shows the most similar trajectories for three query trajectories. For common trajectories (top rows), the most similar trajectories are true near neighbours. It can also be seen that the proposed distance function is rotation invariant. For uncommon trajectories (bottom row), the found candidates are very different from the query. In the remainder we focus on the N-Best algorithm with for a loss-free and exact computation of the top-N matches.

[3] `http://www.orgs.ttu.edu/debs2013`

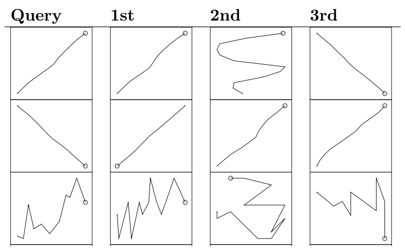


Fig. 2. Most similar trajectories for a given query

5.3 Episode discovery

The first experiments of the episode discovery algorithm focus on the influence of the parameters wrt the number of generated and counted episodes. The algorithm depends on four different parameters, the *similarity*, *frequency*, the *bidirectional evidence*, and the *expiry time*. For this set of experiments, we use the trajectories of team A to find frequent tactical patterns in the game. The results are shown in Figure 3.

The similarity threshold strongly impacts the number of generated episodes: small changes may already lead to an exponential growth in the number of trajectories and large values quickly render the problem infeasible even on medium-sized Hadoop clusters. A similar effect can be observed for the expiry time threshold. Incrementing the expiry time often requires decreasing the similarity threshold. The number of counted episodes is adjusted by the frequency threshold. As shown in the figure, the number of generated episodes can often be reduced by one or more orders of magnitudes. By contrast, the bidirectional evidence threshold affects the result only marginally.

6 Conclusion

We proposed a novel method to mining frequent patterns in positional data streams where consecutive coordinates of objects are treated as movements. We proposed an efficient preprocessing of the positional data using locality sensitive hashing and approximate dynamic time warping and presented a distributed frequent pattern mining algorithm that generalised Achar et al. [1] to positional data at large-scales.

References

1. A. Achar, S. Laxman, R. Viswanathan, and P. S. Sastry. Discovering injective episodes with general partial orders. *Data Min. Knowl. Discov.*, 25(1):67–108, 2012.
2. R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, 1993.

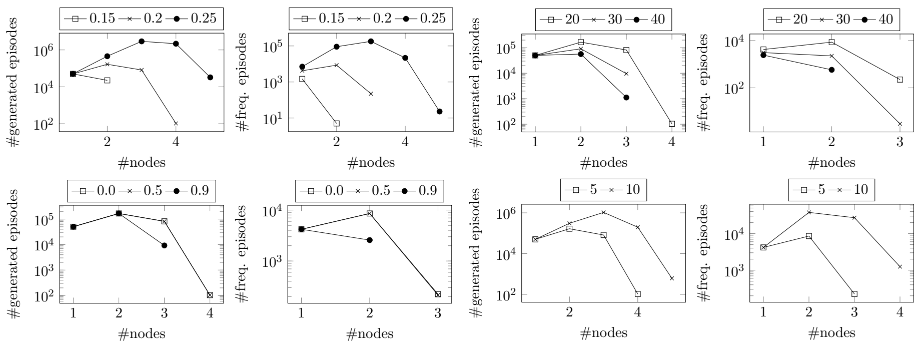


Fig. 3. Top row: Varying similarity (first and second columns) and frequency (third and fourth columns) thresholds. Bottom row: Varying bidirectional evidence (first and second columns) and expiry time (third and fourth columns) thresholds.

3. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of ICDE*, 1995.
4. V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest neighbor retrieval using distance-based hashing. In *Proc. of ICDE*, 2008.
5. F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *Proc. of KDD*, 2007.
6. A. Grunz, D. Memmert, and J. Perl. Tactical pattern recognition in soccer games by means of special self-organizing maps. *Human Movement Science*, 31(2):334–343, 2012.
7. C.-H. Kang, J.-R. Hwang, and K.-J. Li. Trajectory analysis for soccer players. In *Proc. of ICDMW*, 2006.
8. E. Keogh. Exact indexing of dynamic time warping. In *Proc. of VLDB*, 2002.
9. Eamonn J. Keogh and Michael J. Pazzani. Derivative dynamic time warping. In *In First SIAM International Conference on Data Mining (SDM2001*, 2001.
10. S.-W. Kim, S. Park, and W. W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proc. of ICDE*, 2001.
11. S. Laxman, P. S. Sastry, and K. P. Unnikrishnan. Discovering frequent episodes and learning hidden markov models: A formal connection. *IEEE Trans. on Knowl. and Data Eng.*, 17(11):1505–1517, November 2005.
12. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, 1997.
13. M. Perše, M. Kristan, S. Kovačič, G. Vučkovič, and J. Perš. A trajectory-based analysis of coordinated team activity in a basketball game. *Computer Vision and Image Understanding*, 113(5):612 – 621, 2009.
14. L. Rabiner and B.-H. Juang. *Fundamentals of speech recognition.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
15. N. Tatti and B. Cule. Mining closed episodes with simultaneous events. In *Proc. of KDD*, 2011.
16. M. Vlachos, D. Gunopulos, and G. Das. Rotation invariant distance measures for trajectories. In *Proc. of KDD*, 2004.
17. M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Mach. Learn.*, 42(1-2):31–60, 2001.

Semi-supervised learning for multi-target regression

Jurica Levatić[1,2], Michelangelo Ceci[3], Dragi Kocev[1], and Sašo Džeroski[1,2]

[1] Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
[2] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
[3] Department of Computer Science, University of Bari Aldo Moro, Bari, Italy
`name.surname@ijs.si, michelangelo.ceci@uniba.it`

Abstract. The most common machine learning approach is supervised learning, which uses labeled data for building predictive models. However, in many practical problems, the availability of annotated data is limited due to the expensive, tedious and time-consuming annotation procedure. At the same, unlabeled data can be easily available in large amounts. This is especially pronounced for predictive modelling problems with structured output space. Semi-supervised learning (SSL) aims to use unlabeled data as an additional source of information in order to build better predictive models than can be learned from labeled data alone. The majority of work in SSL considers the simple tasks of classification and regression where the output space consists of a single variable. Much less work has been done on SSL for structured output prediction. In this study, we address the task of multi-target regression (MTR), a type of structured output where the output space consists of multiple numerical values. Our main objective is to investigate whether we can improve over supervised methods for MTR by using unlabeled data. We use ensembles of predictive clustering trees in a self-training fashion: most reliable predictions on unlabeled data are iteratively used to re-train the model. We use variance of an ensemble models as an indicator of the reliability of predictions. Our results provide a proof-of-concept: Unlabeled data improves predictive performance of ensembles for multi-target regression, however further efforts are needed to automatically select the optimal threshold for reliability of predictions.

Keywords: semi-supervised learning, self-training, multi-target, multi-output, multivariate, regression, ensembles, structured outputs, PCTs

1 Introduction

The major machine learning paradigms are supervised learning (e.g., classification, regression), where all the data are labeled, and unsupervised learning (e.g., clustering) where all the data are unlabeled. Semi-supervised learning (SSL) [1] examines how to combine both paradigms and exploit both labeled and unlabeled data, aiming to benefit from the information that unlabeled data can bring. SSL is of important practical value since the following scenario can often be encountered: labeled data are scarce and hard to get because they require human experts, expensive devices or time-consuming experiments, while, at the same time, unlabeled data abound and are easily obtainable. Real-world classification problems of this type include: phonetic annotation of human speech, protein 3D structure prediction, and spam filtering. Intuitively, SSL yields best results when there are few labeled examples as compared to unlabeled ones (i.e., large-scale labelling is not affordable). Such a scenario is in particular relevant for machine learning tasks with complex (structured) outputs, where providing the labels of data is a laborious and/or an expensive process, while at the same time large amounts of unlabeled data are readily available.

In this study, we are concerned with the semi-supervised learning for *multi-target regression* (MTR). MTR is a type of structured output prediction task where the goal is to predict multiple continuous target variables (also known as multi-output or multivariate regression). In many real life problems, we are interested in simultaneously predicting multiple continuous variables. Prominent examples come from ecology: predicting abundance of different species living in the same habitat [2], or predicting properties of forest [3]. There are several advantages of learning a multi-target (i.e., global) model over learning a separate (i.e., local) model for each target variable: Global models are typically easier to interpret, perform better and overfit less than collection of single-target models [4]. In the past, classical (single-target) regression received much more attention than MTR, however several researchers proposed methods for solving the task of MTR directly and demonstrated their effectiveness [5–8].

Semi-supervised methods able to solve MTR problems are scarce. Most commonly, SSL methods for structured output prediction are dealing with discrete outputs. Here, prominent work was done by Brefeld [9], who used co-training paradigm and the principle of maximizing the consensus among multiple independent hypotheses to develop semi-supervised support vector learning algorithm for joint input-output spaces and arbitrary loss. Zhang and Yeung [10] proposed a semi-supervised method based on Gaussian processes for a task related to MTR: multi-task regression. In multi-task learning the aim is to predict multiple single-target variables with different training sets (in general, with different descriptive attributes) at the same time. Also related, Navaratnam et al. [11] proposed a semi-supervised method for multivariate regression specialized for computer vision. The goal is to relate features of images (z) to joint angles (θ). Unlabeled examples are used to help the fitting of the joint density $p(z, \theta)$.

In this work, we propose a self-training approach [12] (also called self-teaching or bootstrapping) for performing SSL for MTR. As a base predictive model, we use predictive clustering trees (PCTs), or more precisely, random forest of predictive clustering trees for MTR [8]. PCTs are a generalization of standard decision trees towards predicting structured outputs. They are able to make predictions for several types of structured outputs [8]: tuples of continuous/discrete variables, hierarchies of classes and time series.

The main feature of self-training is that it iteratively uses its own most reliable predictions in the learning process. The most reliable predictions are selected by using a threshold on the reliability scores. The main advantage of the iterative semi-supervised learning approach is that it can be "wrapped" around any existing (supervised) method. The only prerequisite is that the underlying method is able to provide a reliability score for its predictions. With our base predictive models, i.e., random forest of PCTs for MTR, this score is estimated by using the variance of the votes from the ensemble members of an example.

The concept of self-training was first proposed by Yarowsky [13] for word sense disambiguation, i.e., deciding the meaning of a homonym in a given context. Other successful applications of self training include: detection of objects on image [14], identification of subjective nouns [15] and learning human motion over time [16]. There are several examples of methods based on self-training (or based on closely related co-training) implemented for solving the task of (single-target) regression [17–21]. To the best of our knowledge, self-training was not implemented yet for the problem of multi-target regression.

The main purpose of this study is to investigate the following question: Can unlabeled data improve predictive performance of the models for MTR in a self-training setting? To this end, we compared our semi-supervised method to its supervised counterpart in the following evaluation scenario: We consider the best result (considering different thresholds for reliability score) of semi-supervised method. Results show that the proposed semi-supervised method is able to improve over supervised random forest in 4 out of 6 considered datasets. Thus, the evaluation provides a positive answer to our research question posed above, and motivates further research efforts in this direction.

2 Semi-supervised learning with ensembles of PCTs

The basis of the semi-supervised method proposed in this study is the use, in an ensemble learning fashion, of predictive clustering trees (PCTs). In this section, we first briefly describe PCTs for multi-target regression, followed by a description of the method for learning random forest. We then present in details the adaptation of semi-supervised self-training approach for multi-target regression with random forest of PCTs.

2.1 Predictive clustering trees for MTR

The predictive clustering trees framework views a decision tree as a hierarchy of clusters: the top-node corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. The PCT framework is implemented in the CLUS system [22], which is available for download at `http://clus.sourceforge.net`.

PCTs are induced with a standard *top-down induction of decision trees* (TDIDT) algorithm [23] (see Table 1). It takes as input a set of examples (E) and outputs a tree. The heuristic (h) that is used for selecting the tests (t) is the reduction in variance caused by the partitioning ($\mathcal{P}$) of the instances corresponding to the tests (t) (see line 4 of the BestTest procedure in Table 1). By

Table 1. The top-down induction algorithm for PCTs.

```
procedure PCT
Input: A dataset $E$
Output: A predictive clustering tree
1: $(t^*, h^*, \mathcal{P}^*) = \text{BestTest}(E)$
2: if $t^* \neq none$ then
3:     for each $E_i \in \mathcal{P}^*$ do
4:         $tree_i = \text{PCT}(E_i)$
5:     return node$(t^*, \bigcup_i \{tree_i\})$
6: else
7:     return leaf(Prototype$(E)$)
```

```
procedure BestTest
Input: A dataset $E$
Output: the best test $(t^*)$, its heuristic score $(h^*)$ and the partition $(\mathcal{P}^*)$ it induces on the dataset $(E)$
1: $(t^*, h^*, \mathcal{P}^*) = (none, 0, \emptyset)$
2: for each possible test $t$ do
3:     $\mathcal{P}$ = partition induced by $t$ on $E$
4:     $h = Var(E) - \sum_{E_i \in \mathcal{P}} \frac{|E_i|}{|E|} Var(E_i)$
5:     if $(h > h^*) \wedge \text{Acceptable}(t, \mathcal{P})$ then
6:         $(t^*, h^*, \mathcal{P}^*) = (t, h, \mathcal{P})$
7: return $(t^*, h^*, \mathcal{P}^*)$
```

maximizing the variance reduction, the cluster homogeneity is maximized and the predictive performance is improved.

The main difference between the algorithm for learning PCTs and a standard decision tree learner is that the former considers the variance function and the prototype function (that computes a label for each leaf) as *parameters* that can be instantiated for a given learning task. So far, PCTs have been instantiated for the following tasks [8]: multi-target prediction (which includes multi-target regression), hierarchical multi-label classification and prediction of time-series. In this article, we focus on the task of multi-target regression (MTR).

The variance and prototype functions of PCTs for MTR are instantiated as follows. The variance (used in line 4 of the procedure BestTest in Table 1) is calculated as the sum of the variances of the target variables, i.e., $Var(E) = \sum_{i=1}^{T} Var(Y_i)$, where T is the number of target variables, and $Var(Y_i)$ is the variance of target variable Y_i. The variances of the targets are normalized, so each target contributes equally to the overall variance. The normalization is performed by dividing with the estimates with the standard deviation for each target variable on the available training set. The prototype function (calculated at each leaf) returns as a prediction the mean values of the target variables, calculated by using the training instances that belong to the given leaf.

2.2 Ensembles of PCTs

We consider random forest of PCTs for structured prediction, as suggested by Kocev et al. [8] in the CLUS system. The PCTs in the random forest are constructed by using the random forests method given by Breiman [24]. The algorithm of this ensemble learning method is presented in Table 2, left.

A random forest (Table 2, left) is an ensemble of trees, where diversity among the predictors is obtained by using bootstrap replicates and additionally by changing the set of descriptive attributes during learning. Bootstrap samples are obtained by randomly sampling training instances, with replacement, from the original training set, until an equal number of instances as in the training

Table 2. The learning algorithms for random forests and semi-supervised self-training (CLUS-SSL). Here, E_l is the set of the labeled training examples, E_u is a set of unlabeled examples, k is the number of trees in the forest, $f(D)$ is the size of the feature subset considered at each node during tree construction for random forests and τ is the threshold for reliability of predictions.

```
procedure RForest($E_l, k, f(D)$)
returns Forest
1: $F = \emptyset$
2: for $i = 1$ to $k$ do
3:     $E_i = bootstrap(E_l)$
4:     $T_i = PCT\_rnd(E_i, f(D))$
5:     $F = F \bigcup \{T_i\}$
6: return $F$
```

```
procedure CLUS-SSL($E_l$, $E_u$, $\tau$, k, f(D))
returns Forest
1: repeat
2:     $F$ = RForest($E_l, k, f(D)$)
3:     predict($F, E_u$)
4:     for each $e_u \in E_u$ do
5:         if Reliability($F, e_u$) $\geq \tau$ then
6:             move $e_u$ from $E_u$ to $E_l$
7: until No example $e_u$ is moved from $E_u$ to $E_l$
```

set is obtained. Breiman [25] showed that bagging can give substantial gains in predictive performance, when applied to an unstable learner (i.e., a learner for which small changes in the training set result in large changes in the predictions), such as classification and regression tree learners.

To learn a random forest, the classical PCT algorithm for tree construction (Table 1) is replaced by PCT_rnd which replaces the standard selection of attributes with a randomized selection. More precisely, at each node in the decision trees, a random subset of the descriptive attributes is taken, and the best attribute is selected from this subset. The number of attributes that are retained is given by a function f of the total number of descriptive attributes D (e.g., $f(D) = 1$, $f(D) = \lfloor\sqrt{D}+1\rfloor$, $f(D) = \lfloor log_2(D)+1\rfloor$ ...). The reason for random selection of attributes is to avoid (possible) correlation of the trees in a bootstrap sample. Namely, if only few of the descriptive attributes are important for prediction of target variables, these will be selected by many of the bootstrap tress, generating highly correlated trees.

In the random forest of PCTs, the prediction for a new instance is obtained by combining the predictions of all the base predictive models. For the MTR task, the predictions for each target variable is computed as the average of the predictions obtained from each tree.

2.3 Self-training for MTR

To perform semi-supervised learning with ensembles of PCTs for MTR, we consider a self-training approach. In self-training, first a predictive model (i.e., a random forest of PCTs) is constructed by using the available labeled instances. The unlabeled instances are then labeled by using the obtained predictive model. Next, the examples with the most reliable predictions are selected and then added to the training set. A predictive model is again constructed and the procedure is repeated until a stopping criterion is satisfied.

To adapt the self-training procedure to the MTR task, we need to define: *i)* a reliability measure of the predictions, *ii)* a criterion to select the most reliable predictions and *iii)* a stopping criterion. Since self-training relies on the assumptions that its own (most reliable) predictions are correct, the most crucial part of the algorithm is the definition of a good reliability measure. This measure should be able to discern correct (with high reliability score) from wrong (with low reliability score) predictions. At this purpose, we exploit a solution provided directly with the ensemble learning – we use the variance of the votes of an ensemble as an indicator of reliability.

When an unlabeled example is predicted by a random forest, we consider the prediction reliable if predictions of individual trees (i.e., votes) in the ensemble are coherent. Otherwise, if the predictions by individual trees in the ensemble are very heterogeneous, we consider the prediction unreliable. The variance has been previously used in bagging where it has been found to perform the best in an extensive empirical comparison of various approaches for estimating reliability of regression predictions [26].

Here we present the procedure for calculation of reliability score in more detail. Formally, for each iteration of the self-training algorithm, we have to solve an MTR problem with m continuous target variables by learning a random forest ensemble F with k trees. These trees are trained on a set of labeled examples E_l and applied on a set of unlabeled examples E_u. First, for each unlabeled example $e_u \in E_u$, per-target standard deviation of votes of ensemble r_u^i is calculated as:

$$r_u^i = \sqrt{\frac{1}{k-1}\sum_{j=1}^{k}\left(tree_j^i(e_u) - F^i(e_u)\right)^2}, \quad i = 1 \ldots m,$$

where $tree_j^i$ is a vote (i.e., a prediction score) for e_u returned by the j^{th} tree for the i^{th} target. F^i is the prediction for e_u returned by the ensemble for the i^{th} target (i.e., the average of the votes of each tree).

In order to equally weight the contribution of each target attribute in the reliability of the prediction obtained for each unlabeled example, we normalize per-target standard deviations in the interval $[0, 1]$ as follows:

$$\bar{r}_u^i = \frac{r_u^i - \min\limits_{j=1\ldots|E_u|} r_j^i}{\max\limits_{j=1\ldots|E_u|} r_j^i - \min\limits_{j=1\ldots|E_u|} r_j^i}, \quad i = 1 \ldots m.$$

After normalization, the reliability score for an example e_u can be computed by considering the average of the normalized per-target standard deviations:

$$Reliability(F, e_u) = 1 - \frac{1}{m}\sum_{i=1}^{m}\left(\bar{r}_u^i\right)$$

In this formula we have that, a small standard deviation leads to a high score (high reliability).

Table 3. Characteristics of the datasets. N: number of instances, D/C: number of descriptive attributes (discrete/continuous), T: number of target variables.

Dataset	N	D/C	T
Forestry LIDAR IRS [27]	2731	0/29	2
Sigmea real [28]	817	0/4	2
Soil quality [2]	1944	0/142	3
Solar flare-2 [29]	1066	10/0	3
Vegetation clustering [30]	29679	0/65	11
Water quality [31]	1060	0/16	14

The self-training algorithm for MTR with ensembles of PCTs (named CLUS-SSL) is presented in Table 2 (right). To choose which unlabeled examples should be added to the training set we use a user-defined threshold for the reliability score: $\tau \in [0, 1]$. If the reliability of the prediction of an unlabeled example is greater than τ, the example is moved from the unlabeled set (E_u) to the training set (E_l), together with its multi-target predictions. The iterations are repeated until no unlabeled example is moved from the set E_u to the set E_l. This can happen for two reasons, either the set E_u becomes empty, or the reliability score for all the unlabeled examples is smaller than τ.

It is noteworthy that, the combination of random forest and self-training can be considered as a variant of the co-training learning schema where, at each iteration, we do not keep the same views used in the previous iteration and independence among the views is (partially) guaranteed by the ensemble learning approach. This guarantees that the semi-supervised approach can still improve prediction even if, at each iteration, it considers the same features.

3 Experimental design

The semi-supervised method for MTR proposed in this study (CLUS-SSL) iteratively trains random forest tree ensemble for MTR. Thus, we compare the predictive performance of the CLUS-SSL to the performance of a supervised random forest, which is considered as baseline for comparison. The exact evaluation procedure is presented in more details in the remainder of this section.

3.1 Data description

To evaluate the predictive performance of the methods, we use six dataset with multiple continuous target variables. The selected datasets are mainly from the domain of ecological modelling. The main characteristics of the datasets are provided in Table 3. We can observe that the datasets vary in the size, number of attributes and number of target variables.

3.2 Experimental setup and evaluation procedure

Random forests used in the experimental evaluation were constructed with 100 trees. Trees were not pruned and the number of random features used in random forest was set to $\lfloor \log_2(D) + 1 \rfloor$, where D is the total number of features, as recommended by Breiman [24].

To evaluate the predictive performance of the models, we use a procedure similar to 5-fold cross validation, with the difference that the training folds are further partitioned into labeled and unlabeled. More specifically, the data are first randomly divided into 5 folds. Each fold is used once as a test set, and the remaining four folds are used for training. From the training folds, we randomly chose a percentage of the data which serve as labeled examples. We remove the labels of other examples and provide them to the algorithm to serve as unlabeled data during training. Supervised random forests were trained only on the labeled part of the data. The predictive performance reported in the results is the average obtained on the 5 test sets.

To investigate the influence of the amount of labeled data, for each dataset we vary the ratio of labeled versus unlabeled data, where percentage of labeled relative to unlabeled data ranges in the following set: [1%, 3%, 5%, 7%, 10%, 15%, 20%, 30%, 50%].

For the CLUS-SSL algorithm, we need to set the threshold τ for the reliability score, which is used throughout the iterations. For each percentage of labeled data, we tested 15 different thresholds:

$$\tau = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.99\}.$$

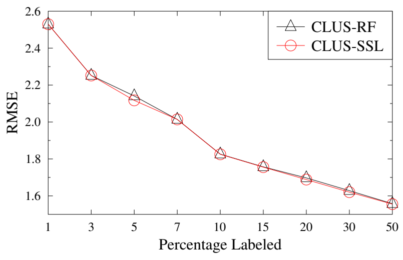
Therefore, 15 predictive models were built (one model corresponding to one threshold) for each percentage of labeled data. Among these, we report the predictive performance of the best model.

We evaluate the algorithms by using the *root mean square error* (RMSE):

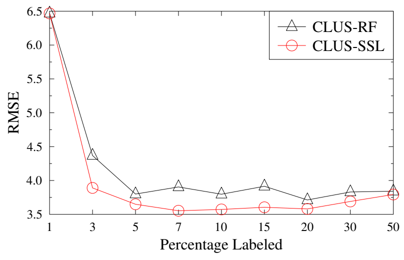
$$RMSE = \sqrt{1/(N * m) * \sum_{i=1}^{m} \sum_{j=1}^{N} \left(a_j^i - p_j^i\right)^2},$$

where m is the number of target variables, N is the number of examples, a_j^i is the real value of the i^{th} target of the j^{th} example, and p_j^i is the predicted value of the i^{th} target of the j^{th} example.

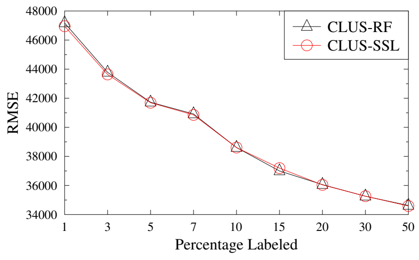
In order to make results comparable across different percentages of labeled examples, we opted to use an evaluation procedure where the test sets are consistent for all the settings. In the results reported in this paper, we consider that the optimal threshold is provided by an 'oracle'. Such threshold selection procedure suffices for answering the research question investigated in this work: *Can unlabeled data potentially improve the predictive performance of models for MTR?* A more general solution for selecting the threshold, would be to use a cross-validation procedure or by implementing smarter thresholding system in self-training which tries to automatically detect the optimal threshold. This aspect is left as future work.



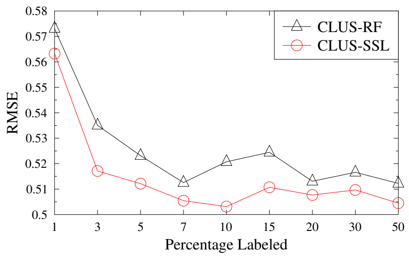
(a) Forestry LIDAR IRS



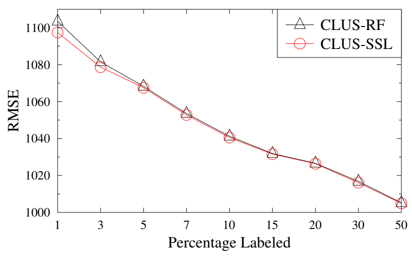
(b) Sigmea real



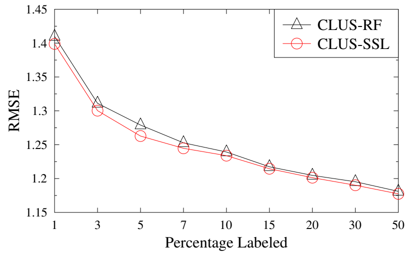
(c) Soil quality



(d) Solar flare-2



(e) Vegetation clustering



(f) Water quality

Fig. 1. Comparison of predictive performance of random forest (CLUS-RF) and semi-supervised self-training (CLUS-SSL). Percentage of labeled data varies from 1% to 50%. For each pertentage of labeled data, the best result for CLUS-SSL is presented, considering the different thresholds for confidence of predictions. Optimal threshold is indicated on the plot. CLUS-SSL performs very similar to CLUS-RF (a and c) or improves over CLUS-RF (b, d, e and f).

4 Results and discussion

The results of the experimental evaluation are presented in Figure 1. Their analysis reveals that the proposed semi-supervised method (CLUS-SSL) outperforms its supervised counterpart (CLUS-RF) on 4 out of 6 datasets: Sigmea real, Solar flare-2, Water quality and Vegetation clustering. On the other two datasets (Forestry LIDAR IRS and Soil quality), the two methods perform very similar, with small improvements or degradations in performance made by CLUS-SSL. It was noted before that the success of SSL is domain depended, i.e., methods can behave very differently depending on the nature of the datasets, and that no single SSL method consistently performs better than supervised learning [32].

Table 4. Optimal threshold for reliability of predictions (τ), the percentage of unlabeled examples added to the training set after the completion of the self-training procedure ($\mathcal{E}$) and the number of iterations performed ($\mathcal{I}$) of the CLUS-SSL method.

Dataset		Percentage of labeled data								
		1%	3%	5%	7%	10%	15%	20%	30%	50%
Forestry LIDAR IRS	τ	0.99	0.99	0.9	0.99	0.95	0.95	0.9	0.95	0.95
	$\mathcal{E}$	0%	0%	28%	0%	5%	3%	26%	3%	2%
	$\mathcal{I}$	1	1	80.8	1	26.4	18	57.4	14	8.2
Sigmea real	τ	0.4	0.85	0.6	0.95	0.85	0.55	0.7	0.8	0.65
	$\mathcal{E}$	100%	100%	100%	100%	100%	100%	100%	99%	100%
	$\mathcal{I}$	6.8	9.8	7	16	8.8	6.4	8	5.6	5.2
Soil quality	τ	0.9	0.95	0.7	0.9	0.99	0.95	0.9	0.99	0.95
	$\mathcal{E}$	4%	1%	95%	26%	0%	2%	34%	0%	4%
	$\mathcal{I}$	3.8	2.6	8.8	15	1	2.4	11	1	5.2
Solar flare-2	τ	0.9	0.7	0.8	0.55	0.65	0.75	0.75	0.55	0.9
	$\mathcal{E}$	99%	98%	91%	100%	100%	97%	96%	100%	83%
	$\mathcal{I}$	15.2	5	11	4	5.4	9	7	4.4	9.2
Vegetation clustering	τ	0.1	0.5	0.4	0.95	0.9	0.85	0.9	0.9	0.95
	$\mathcal{E}$	100%	100%	100%	0%	1%	7%	2%	2%	0%
	$\mathcal{I}$	2	7.4	4.6	1.2	32.8	82.6	43.6	52.4	7.2
Water quality	τ	0.3	0.65	0.65	0.5	0.4	0.65	0.5	0.4	0.55
	$\mathcal{E}$	100%	100%	100%	100%	100%	99%	100%	100%	99%
	$\mathcal{I}$	3.2	36.2	32.2	8	3.8	29.8	8.2	4.2	16.8

Results reported in this paper are, thus, consistent with results obtained in previous research on tasks which are different from MTR.

The analysis of the results by varying the percentage of labeled data shows that, as expected, RMSE error decreases with the increase of the percentage of labeled data used to construct the predictive model (better models are learned with more data). However, these trends are not observed across all of the datasets. We can observe the saturation in performance for Sigmea real and Solar flare-2 datasets. There, from about 5% to 7% percent of labeled data, both methods (CLUS-SSL and CLUS-RF) were not able to improve much in the absolute terms. In spite of that, CLUS-SSL is consistently performing better than CLUS-RF, meaning that even in situations where supervised models reached saturation, unlabeled data can further boost the performance. On the other two datasets where unlabeled data helps (Vegetation clustering and Water quality), the improvements of CLUS-SSL over CLUS-RF are more notable with smaller percentages of labeled data. Such behavior is expected, since SSL has the best potential when not much labeled examples are available.

In Table 4, the specific conditions used to obtain and evaluate the CLUS-SSL models (whose performances are depicted in Fig. 1) are given. When observing the variability of the optimal thresholds for the reliability score, we cannot detect regularities. They vary greatly from one dataset to another, and from one percentage of labeled data to another, meaning that it is hard to tell in advance

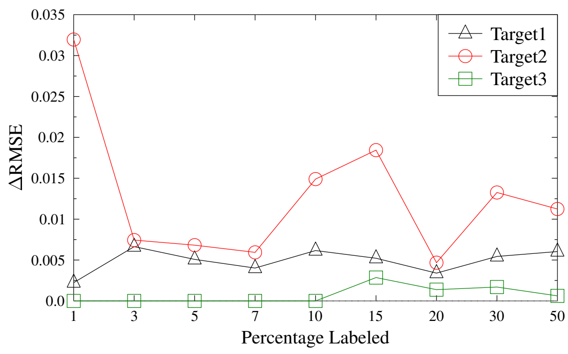


Fig. 2. Analysis of per target performance for the Solar flare-2 dataset, in terms of difference in performance between CLUS-RF and CLUS-SSL (ΔRMSE). Positive values suggest that CLUS-SSL is better, while negative that CLUS-RF is better. Zero means that there is no difference in performance.

which threshold should be used. Self-training can also degrade performance of the underlying method if a sub-optimal threshold is chosen. In particular, if a too permissive threshold is selected, it can allow wrongly predicted examples to enter in the training set. A classification error in the earliest iterations can reinforce itself in the next iterations, leading to a degradation of the performance. On the other hand, if a too stringent threshold is set, it is possible that none, or very few, of the unlabeled examples will enter the training set, meaning that we will miss the opportunity to improve performance with unlabeled data.

Similar observation can be made for the number of performed self-training iterations, they are very heterogeneous regarding the different datasets and percentages of labeled data. Analysis of the number of unlabeled examples added to the training set reveals that, in the cases where semi-supervised learning helps, almost all of the unlabeled examples were moved to the training set at the end of the self-training procedure. This is very consistent across datasets where CLUS-SSL improves over CLUS-RF: Sigmea real, Solar flare-2, Vegetation clustering (for the cases with 1 to 5% percent of labeled data) and Water Quality. The fundamental assumption of self-training is that its most reliable predictions are correct. Thus, the success of this method depends on the ability to learn an accurate model from the data at hand. The assumption is apparently met on the former four cases. Moreover, the (good) predictive ability of the models was retained throughout iterations, as all unlabeled examples were eventually added to the training set. Contrary, if CLUS-SSL was not able to improve over CLUS-RF, then generally very few, or none of the unlabeled examples were added to the training set. This is the case at Forestry LIDAR IRS, Soil Quality and Vegetation Clustering (for more than 5% of labeled data) datasets. The predictive models learnt from these datasets are most probably prone to errors and the self-training approach would only lead to a propagation of the errors (this is confirmed by the optimal threshold for reliability close to 1).

A different perspective of the results is provided in Figure 2, where per-target RMSE improvements over the baseline are shown. As it is possible to see, these results show that the improvement provided by the semi-supervised setting is not uniform over the different targets. This means that for some target attributes, there is still a large margin for improvement with respect to accuracy reached by the random forest approach.

5 Conclusions and further work

Semi-supervised learning is an intriguing research area because of potential gains in performance for 'free' – labeling of the data is expensive and laborious, while freely available unlabeled data can be used to enhance the performance of traditional, supervised, machine learning methods. Such proposition is even more relevant for learning problems with structured outputs, where labeling of the data is even more expensive and problematic.

We address the task of semi-supervised learning for multi-target regression – a type of structured output, where the goal is to simultaneously predict multiple continuous variables. To the best of our knowledge, semi-supervised methods dealing with this task do not exist thus far. We propose a self-training approach to semi-supervised learning by using a random forest of predictive clustering trees for multi-target regression. In the proposed approach, a model uses its own most reliable predictions in an iterative fashion.

Due to its relative simplicity and intuitiveness, self-training can be considered as a baseline semi-supervised approach, i.e., a starting point for investigation of the influence of unlabeled data. In this study, we wanted to investigate whether unlabeled data can improve predictive performance of the models for MTR in a self-training setting. The results of the experimental evaluation show that the proposed method outperforms its supervised counterpart on 4 out of 6 datasets. These are encouraging results and prompt further investigation.

In future, we plan to extend this work along several directions. First, we plan to implement an intelligent threshold selection procedure. Namely, here we consider a relatively simple implementation of self-training (with respect to the thresholding system and the stopping criteria), but there are several possibilities to implement more sophisticated variants of it. For instance, so-called 'airbag' stopping criteria [33] can detect degradation in performance and stop self-training. Alternatively, we can utilize 'out-of-bag properties' of the random forest to automatically detect the optimal threshold for the reliability score. Second, success of the reliability estimate of regression predictions can vary depending on the domain or the regression model used. The most appropriate estimates can be automatically detected [34] and used during self-training. Third, modularity of predictive clustering trees enables easy extension of the self-training approach to the other types of structured outputs, such as multi-target classification or time-series prediction.

Acknowledgments

We would like to acknowledge the support of the European Commission through the project MAESTRA - Learning from Massive, Incompletely annotated, and Structured Data (Grant number ICT-2013-612944).

References

1. Chapelle, O., Schölkopf, B., Zien, A.: Semi-supervised Learning. Volume 2. MIT Press, Cambridge, MA (2006)
2. Demšar, D., Džeroski, S., Larsen, T., Struyf, J., Axelsen, J., Pedersen, M., Krogh, P.: Using multi-objective classification to model communities of soil. Ecological Modelling **191**(1) (2006) 131–143
3. Stojanova, D., Panov, P., Gjorgjioski, V., Kobler, A., Džeroski, S.: Estimating vegetation height and canopy cover from remotely sensed data with machine learning. Ecological Informatics **5**(4) (2010) 256–266
4. Levatić, J., Kocev, D., Džeroski, S.: The use of the label hierarchy in hierarchical multi-label classification improves performance. In Appice, A., et al., eds.: New Frontiers in Mining Complex Patterns. Volume 8399 of Lecture Notes in Computer Science. Springer International Publishing, Switzerland (2014) 1–16
5. Appice, A., Džeroski, S.: Stepwise induction of multi-target model trees. In Kok, J.N., Koronacki, J., Mantaras, R.L.d., Matwin, S., Mladenić, D., Skowron, A., eds.: Machine Learning: ECML 2007. Volume 4701 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2007) 502–509
6. Struyf, J., Džeroski, S.: Constraint based induction of multi-objective regression trees. In: Proceedings of the 4th International Workshop on Knowledge Discovery in Inductive Databases, LNCS 3933, Springer, Berlin (2006) 222–233
7. Kocev, D., Džeroski, S., White, M.D., Newell, G.R., Griffioen, P.: Using single- and multi-target regression trees and ensembles to model a compound index of vegetation condition. Ecological Modelling **220**(8) 1159–1168
8. Kocev, D., Vens, C., Struyf, J., Džeroski, S.: Tree ensembles for predicting structured outputs. Pattern Recognition **46**(3) (2013) 817–833
9. Brefeld, U.: Semi-supervised Structured Prediction Models. PhD thesis, Humboldt-Universität zu Berlin, Berlin, Germany (2008)
10. Zhang, Y., Yeung, D.Y.: Semi-supervised multi-task regression. In Buntine, W., Grobelnik, M., Mladeni, D., Shawe-Taylor, J., eds.: Machine Learning and Knowledge Discovery in Databases. Volume 5782 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2009) 617–631
11. Navaratnam, R., Fitzgibbon, A., Cipolla, R.: The joint manifold model for semi-supervised multi-valued regression. In: Proceedings of the 11th IEEE International Conference on Computer Vision. (2007) 1–8
12. Zhu, X.: Semi-supervised learning literature survey. Technical report, Computer Sciences, University of Wisconsin-Madison (2008)
13. Yarowsky, D.: Unsupervised word sense disambiguation rivaling supervised methods. In: Proceedings of the 33rd annual meeting on Association for Computational Linguistics. (1995) 189–196
14. Rosenberg, C., Hebert, M., Schneiderman, H.: Semi-supervised self-training of object detection models. In: Proceedings of the 7th IEEE Workshop on Applications of Computer Vision. (2005)
15. Riloff, E., Wiebe, J., Wilson, T.: Learning subjective nouns using extraction pattern bootstrapping. In: Proceedings of the 7th Conference on Natural Language Learning. (2003) 25–32

16. Bandouch, J., Jenkins, O.C., Beetz, M.: A self-training approach for visual tracking and recognition of complex human activity patterns. International Journal of Computer Vision **99**(2) (2012) 166–189
17. Brefeld, U., Grtner, T., Scheffer, T., Wrobel, S.: Efficient co-regularised least squares regression. In: Proceedings of the 23rd international conference on Machine learning. (2006) 137–144
18. Zhou, Z.H., Li, M.: Semi-supervised regression with co-training style algorithms. IEEE Transaction in Knowledge Data Engineering **19**(11) (2007) 1479–1493
19. Appice, A., Ceci, M., Malerba, D.: An iterative learning algorithm for within-network regression in the transductive setting. In Gama, J., Costa, V., Jorge, A., Brazdil, P., eds.: Discovery Science. Volume 5808 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2009) 36–50
20. Appice, A., Ceci, M., Malerba, D.: Transductive learning for spatial regression with co-training. In: Proceedings of the 2010 ACM Symposium on Applied Computing. (2010) 1065–1070
21. Yang, M.C., Wang, Y.C.F.: A self-learning approach to single image super-resolution. IEEE Transactions on Multimedia **15**(3) (2013) 498–508
22. Blockeel, H., Struyf, J.: Efficient algorithms for decision tree cross-validation. Journal of Machine Learning Research **3** (2002) 621–650
23. Breiman, L., Friedman, J., Olshen, R., Stone, C.J.: Classification and Regression Trees. Chapman & Hall/CRC (1984)
24. Breiman, L.: Random forests. Machine Learning **45**(1) (2001) 5–32
25. Breiman, L.: Bagging predictors. Machine Learning **24**(2) (1996) 123–140
26. Bosnić, Z., Kononenko, I.: Comparison of approaches for estimating reliability of individual regression predictions. Data & Knowledge Engineering **67**(3) (2008) 504–516
27. Stojanova, D.: Estimating forest properties from remotely sensed data by using machine learning. Master's thesis, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia (2009)
28. Demšar, D., Debeljak, M., Lavigne, C., Džeroski, S.: Modelling pollen dispersal of genetically modified oilseed rape within the field. In: The Annual Meeting of the Ecological Society of America. (2005)
29. Asuncion, A., Newman, D.: UCI machine learning repository (2007)
30. Gjorgjioski, V., Džeroski, S.: Clustering analysis of vegetation data. Technical report, Jožef Stefan Institute (2003)
31. Blockeel, H., Džeroski, S., Grbović, J.: Simultaneous prediction of multiple chemical parameters of river water quality with tilde. In: Proceedings of the 3rd European Conference on PKDD. Volume 1704 of LNAI. (1999) 32–40
32. Chawla, N., Karakoulas, G.: Learning from labeled and unlabeled data: An empirical study across techniques and domains. Journal of Artificial Intelligence Research **23**(1) (2005) 331–366
33. Leistner, C., Saffari, A., Santner, J., Bischof, H.: Semi-supervised random forests. In: Proceedings of the 12th International Conference on Computer Vision. (2009) 506–513
34. Bosnić, Z., Kononenko, I.: Automatic selection of reliability estimates for individual regression predictions. The Knowledge Engineering Review **25**(1) (2010) 27–47

Evaluation of Different Data-derived Label Hierarchies in Multi-label Classification

Gjorgji Madjarov, Tomche Delev, Ivica Dimitrovski, and Dejan Gjorgjevikj

Ss. Cyril and Methodius University, Faculty of Computer Science and Engineering, Rudgjer Boshkovikj 16, 1000 Skopje, Macedonia
{gjorgji.madjarov, tomche.delev, ivica.dimitrovski, dejan.gjorgjevikj}@finki.ukim.mk

Abstract. Motivated by an increasing number of new applications, the research community is devoting an increasing amount of attention to the task of multi-label classification (MLC). Many different approaches to solving multi-label classification problems have been recently developed. Recent empirical studies have comprehensively evaluated many of these approaches on many datasets using different evaluation measures. The studies have indicated that the predictive performance and efficiency of the approaches could be improved by using data derived (artificial) hierarchies, in the learning and prediction phases. In this paper, we compare different clustering algorithms for constructing the label hierarchies (in a data-driven manner), in multi-label classification. We consider flat label sets and construct the label hierarchies from the label sets that appear in the annotations of the training data by using four different clustering algorithms (balanced k-means, agglomerative clustering with single and complete linkage and predictive clustering trees). The hierarchies are then used in conjunction with global hierarchical multi-label classification (HMC) approaches.

Keywords: multi-label, hierarchical, classification, clustering

1 Introduction

Multi-label learning is concerned with learning from examples, where each example is associated with multiple labels. Multi-label classification (MLC) has received significant attention in the research community over the past few years, motivated by an increasing number of new applications. The latter include semantic annotation of images and video (news clips, movies clips), functional genomics (predicting gene and protein function), music categorization into emotions, text classification (news articles, web pages, patents, e-mails, bookmarks...), directed marketing and others.

Madjarov et al. [1] presented an extensive experimental evaluation of the most popular methods for multi-label learning using a wide range of evaluation measures on a variety of datasets. In particular, the authors have experimentally evaluated 12 methods using 16 evaluation measures over 11 benchmark datasets. The results reveal that the best performing methods over all evaluation measures are the Hierarchy Of Multi-label classifiERs (HOMER) [2] and Random Forests of Predictive Clustering Trees for Multi-target Classification (RF-PCTs for MTC) [3], followed by Binary Relevance (BR) [4] and Classifier Chains (CC) [5].

We believe that the better predictive performance and efficiency of the HOMER method as compared to BR and CC, is a result of the data derived (artificial) hierarchy, that HOMER defines over the output space of the original MLC problem first, and then uses it in the learning and prediction phases. In particular, HOMER transforms the (original, flat) multi-label learning task into a hierarchy of (simpler) multi-label learning tasks, based on a hierarchy of labels derived from the data. The hierarchy is obtained by applying an unsupervised (clustering) approach to the label part of the data that comes from the original MLC problem. An example hierarchy of labels (and classifiers) produced for a multi-label classification task with 8 labels $\{\lambda_1, \lambda_2, ..., \lambda_8\}$ is given in Figure 1.

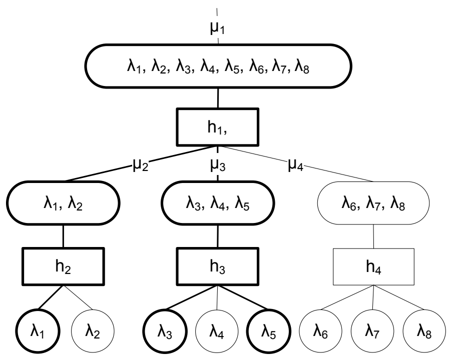


Fig. 1. An example of labels and classifiers considered by HOMER (λ - label, μ - meta-label, h - multi-label classifier).

In this paper, we experimentally evaluate the influence of different data-derived label hierarchies on the predictive performance of multi-label classifiers. Additionally, we confirmed even stronger, that structuring the output space (label part) of a flat MLC problem, and using this structure by a classifier that can directly handle hierarchical multi-label classification (HMC) problems can improve the predictive performance of a classifier that does not use this structure and directly solves the flat MLC problems. More specifically, we derive a hierarchy from the output space of the (original) flat MLC problem using four different clustering approaches first, and then use PCTs for HMC [6] for solving the newly defined hierarchical multi-label classification problem.

To show the improvements that can be achieved by using the data derived structure on the label space and to evaluate the influence of the different data-derived label hierarchies in multi-label classification, we compare: single PCT [6] for solving classical MLC problems [3], and single PCT for solving HMC problems [7] (both in global settings). The transformation of the (original) flat MLC problem to HMC problem is made by balanced k-means clustering [2], agglomerative clustering with single and complete linkage [8] and clustering performed by predictive clustering trees for multi-target classification (MTP) [6].

The remainder of this paper is organized as follows. Section 2 defines the tasks of multi-label classification, multi-label ranking and hierarchical multi-label classification. The use of data derived label hierarchies in multi-label classification is presented in Section 3. Section 4 describes the multi-label datasets, the evaluation measures and the experimental setup, while Section 5 presents and discusses the experimental results. Finally, the conclusions and directions for further work are presented in Section 6.

2 Background

In this section, we define the task of multi-label classification and the task of hierarchical multi-label classification.

2.1 The task of multi-label classification (MLC)

Multi-label learning is concerned with learning from examples, where each example is associated with multiple labels. These multiple labels belong to a predefined set of labels. We can distinguish two types of tasks: multi-label classification and multi-label ranking.

In the case of multi-label classification, the goal is to construct a predictive model that will provide a list of relevant labels for a given, previously unseen example. On the other hand, the goal of the task of multi-label ranking is to construct a predictive model that will provide, for each unseen example, a list of preferences (i.e., a ranking) on the labels from the set of possible labels.

The task of multi-label learning is defined as follows [9]:
Given:

- An input space $\mathcal{X}$ that consists of vectors of values of primitive data types (nominal or numeric), i.e., $\forall \mathbf{x_i} \in \mathcal{X}, \mathbf{x_i} = (x_{i_1}, x_{i_2}, ..., x_{i_D})$, where D is the size of the vector (or number of descriptive attributes),
- an output space $\mathcal{Y}$ that is defined as a subset of a finite set of disjoint labels $\mathcal{L} = \{\lambda_1, \lambda_2, ..., \lambda_Q\}$ ($Q > 1$ and $\mathcal{Y} \subseteq \mathcal{L}$)
- a set of examples E, where each example is a pair of a vector and a set from the input and output space respectively, i.e., $E = \{(\mathbf{x_i}, \mathcal{Y}_i) | \mathbf{x_i} \in \mathcal{X}, \mathcal{Y}_i \subset \mathcal{L}, 1 \leq i \leq N\}$ where N is the number of examples of E ($N = |E|$), and
- a quality criterion q, which rewards models with high predictive performance and low computational complexity.

If the task at hand is multi-label classification, then the goal is to
Find: a function h: $\mathcal{X} \rightarrow 2^{\mathcal{L}}$ such that h maximizes q.

On the other hand, if the task is multi-label ranking, then the goal is to **Find:** a function $f\colon \mathcal{X} \times \mathcal{L} \to \mathcal{R}$, such that f maximizes q, where $\mathcal{R}$ is the ranking on the labels for a given example.

An extensive bibliography of learning methods for solving multi-label learning problems can be found in [10] [4] [11] [1].

2.2 The task of hierarchical multi-label classification (HMC)

Hierarchical classification differs from the multi-label classification in the following: the labels are organized in a hierarchy. An example that is labeled with a given label is automatically labeled with all its parent-labels (this is known as the hierarchy constraint). Furthermore, an example can be labeled simultaneously with multiple labels that can follow multiple paths from the root label. This task is called hierarchical multi-label classification (HMC).

Here, the output space $\mathcal{Y}$ is defined with a label hierarchy $(\mathcal{L}, \leq_h)$, where $\mathcal{L}$ is a set of labels and $\leq_h$ is a partial order representing the parent-child relationship $(\forall\ \lambda_1, \lambda_2 \in \mathcal{L} : \lambda_1 \leq_h \lambda_2$ if and only if λ_1 is a parent of $\lambda_2)$ structured as a tree [9]. Each example from the set of examples E is a pair of a vector and a set from the input and output space respectively, where the set satisfies the hierarchy constraint, i.e., $E = \{(\mathbf{x_i}, \mathcal{Y}_i) | \mathbf{x_i} \in \mathcal{X}, \mathcal{Y}_i \subseteq \mathcal{L}, \lambda \in \mathcal{Y}_i \Rightarrow \forall \lambda' \leq_h \lambda : \lambda' \in \mathcal{Y}_i, 1 \leq i \leq N\}$ where N is the number of examples of E $(N = |E|)$. The quality criterion q, rewards models with high predictive performance and low complexity as in the task of multi-label classification.

An extensive bibliography of learning methods for hierarchical classification scattered across different application domains is given by [12].

3 The use of data derived label hierarchies in multi-label classification

In this study, we suggest to transform the flat multi-label classification problem into a hierarchical multi-label one and solve it by using an approach for HMC [12]. In particular, one should derive a hierarchy from the label part of the original (flat) multi-label classification problem first, and then use this hierarchy to construct hierarchical classification problem that later solves by using a HMC approach [12].

Table 1 shows an example of a multi-label dataset and its corresponding transformed hierarchical multi-label dataset. The transformation is performed according to the label hierarchy generated by a clustering algorithm that considers only the label part (output space) of the training data. In particular, the third column (*Original label set*) in Table 1 shows the labels of the (original) label space of a multi-label learning dataset with five examples. It is defined over a set of eight labels $(\mathcal{L} = \{\lambda_1, \lambda_2, ..., \lambda_8\})$. The fourth column in the same table (*Hierarchical label set*), shows the corresponding hierarchical label set (for the same dataset), obtained by using the label hierarchy from Figure 1 $(\mathcal{HL} = \{\mu_1, \mu_2, \mu_3, \mu_4, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7, \lambda_8\})$. Each example in the HMC dataset is actually labeled with multiple paths of the hierarchy, defined from the root to the leaves (represented by the relevant labels for the corresponding example in the original MLC dataset).

Table 1. A hierarchical multi-label dataset obtained by transforming an original flat multi-label dataset (the label hierarchy from Figure 1 is used)

Example	Features	Original label set	Hierarchical label set
$\mathbf{x}_1$	$x_{11}, x_{12}, \ldots, x_{1D}$	$\{\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_4, \boldsymbol{\lambda}_8\}$	$\{\mu_1, \mu_2, \mu_3, \mu_4, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_4, \boldsymbol{\lambda}_8\}$
$\mathbf{x}_2$	$x_{21}, x_{22}, \ldots, x_{2D}$	$\{\boldsymbol{\lambda}_3, \boldsymbol{\lambda}_6\}$	$\{\mu_1, \mu_3, \mu_4, \boldsymbol{\lambda}_3, \boldsymbol{\lambda}_6\}$
$\mathbf{x}_3$	$x_{31}, x_{32}, \ldots, x_{3D}$	$\{\boldsymbol{\lambda}_1\}$	$\{\mu_1, \mu_2, \boldsymbol{\lambda}_1\}$
$\mathbf{x}_4$	$x_{41}, x_{42}, \ldots, x_{4D}$	$\{\boldsymbol{\lambda}_2, \boldsymbol{\lambda}_3, \boldsymbol{\lambda}_4, \boldsymbol{\lambda}_8\}$	$\{\mu_1, \mu_2, \mu_3, \mu_4, \boldsymbol{\lambda}_2, \boldsymbol{\lambda}_3, \boldsymbol{\lambda}_4, \boldsymbol{\lambda}_8\}$
$\mathbf{x}_5$	$x_{51}, x_{52}, \ldots, x_{5D}$	$\{\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_4, \boldsymbol{\lambda}_7\}$	$\{\mu_1, \mu_2, \mu_3, \mu_4, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_4, \boldsymbol{\lambda}_7\}$

3.1 Generating a label hierarchy on a multi-label output space

The process of generating label hierarchies on a multi-label output space is critical for the good performance of the HMC methods on the transformed problems. When we build the hierarchy over the label space, there is only one constraint that we should take care of: the original MLC task should be defined by the leaves of the label hierarchy. In particular, the labels from the original MLC problem represent the leaves of the tree hierarchy (Figure 1), while the labels that represent the internal nodes of the tree hierarchy are so-called meta-labels (that model the correlation among the original labels).

In this study, we use four different clustering approaches (two divisive and two agglomerative) for deriving the hierarchy on the output space of the (original) MLC problem:

– balanced k-means clustering approach [2] (divisive approach),
– predictive clustering trees [6] (divisive approach),
– agglomerative clustering by using complete linkage [8], and
– agglomerative clustering by using single linkage [8].

Balanced k-means creates the label hierarchy by partitioning the original labels recursively in a top-down depth-first fashion. The top node of the hierarchy contains all labels. At each node n, $k <= |\mathcal{L}_n|$ child nodes are created. The labels of the current node are distributed (divided) using a clustering method into k disjoint subsets (k meta-labels) with an explicit constraint on the size of each subset, one for each child of the current node.

In this work, we use a specific setting from the predictive clustering framework as in [13] [3], where the target space is equal to the descriptive space, i.e., the descriptive variables are used to provide descriptions for the obtained clusters. This focuses the predictive clustering setting on the task of clustering instead of classification.

Agglomerative clustering algorithms treat each example as a singleton cluster at the outset and then successively merge pairs of clusters until all clusters have been merged into a single cluster that contains all examples.

The predictive clustering trees and the agglomerative approaches produce binary tree hierarchies, while the balanced k-means clustering approach produces multi-branch tree hierarchies for $k > 2$.

3.2 Solving MLC problems by using classification approaches for HMC

After the transformation of the original MLC problem into a HMC one, the new HMC problem can be solved by a hierarchical multi-label learning approach. The transformed hierarchical multi-label dataset satisfies the hierarchy constraint (an example that is labeled with a given label is automatically labeled with all its parent-labels).

Figure 2 presents the pseudo-code of the algorithm for solving a MLC problem by using data-derived label hierarchies and a classification approach for HMC. The algorithm first defines the hierarchy, then solves the HMC problem by using a classification approach for HMC. It finally extracts the predictions for the leaves of the hierarchy (that are actually the predictions for the original labels) and evaluates the performance.

E^{train} and E^{test} denote the training and testing examples, while $\mathbf{W}^{train}$ is only the label part (label data) of the training set. Using the label hierarchy derived from the label data, $\mathbf{W}^{train}$ is transformed into new hierarchically organized label data $\mathbf{W}_H^{train}$. E_H^{train} and E_H^{test} denote the corresponding hierarchical multi-label datasets obtained by transforming the original (flat) multi-label datasets (E^{train} and E^{test}) into hierarchical form.

P_H denotes the predictions for the examples of the hierarchical multi-label dataset E_H^{test}, while P denotes the predictions for the original labels. The latter are obtained by extracting the probabilities in the leaves of the label tree from the predictions P_H. The predictions P_H are represented as vectors of probabilities (one vector for one example), where each probability is associated to only one label from the hierarchy (meta-label representing an internal node or original label representing a leaf). Predictions P in the original multi-label scenario can be obtained by using different approaches for transforming the hierarchical multi-label predictions P_H. In this work, we use the simplest approach: only the probabilities for the leaves from the hierarchical predictions P_H are evaluated, while the other probabilities (for the meta-labels) are simply ignored.

3.3 Classification approaches for HMC

Based on the existing literature, Silla et al. [12] propose a unifying framework for hierarchical classification, including a taxonomy of hierarchical classification problems and methods. One of the dimensions along which the hierarchical classification methods differ is the way of using (exploring) the hierarchical label

```
procedure MLCToHMC($E^{train}$ ,$E^{test}$) returns performance
 1: $\mathbf{W}^{train}$ = ExtractLabelSet($E^{train}$);
 2: $\mathbf{W}_H^{train}$ = DefineHierarchy($\mathbf{W}^{train}$);
 3:
 4: //transform multi-label dataset to hierarchical multi-label one
 5: $E_H^{train}$ = MLCToHMCTrainDataset($E^{train}$, $\mathbf{W}_H^{train}$);
 6: $E_H^{test}$ = MLCToHMCTestDataset($E^{test}$, $\mathbf{W}_H^{train}$);
 7:
 8: //solve transformed hierarchical multi-label problem
 9: //by using approach for HMC
10: HMCModel = HMCMetod($E_H^{train}$);
11:
12: //generate HMC predictions
13: $P_H$ = HMCModel($E_H^{test}$);
14:
15: //Extract predictions only for the leaves from the HMC predictions $P_H$
16: $P$ = ExtractLeavesPredictionsFromHMCPredictions($P_H$, $\mathbf{W}_H^{train}$, $\mathbf{W}^{train}$);
17: return EvaluatePredictions(P);
```

Fig. 2. Solving flat MLC problems by using classification approaches for HMC.

structure in the learning and prediction phases. They reviewed two different approaches that utilize the hierarchy: the top-down (or local) approach that uses local information to create a set of local classifiers and the global (or big-bang) approach.

The recent research show that learning a single global model for all labels (in the hierarchy) can have some advantages [3] [14] over the local approaches. The total size of the global classification model is typically smaller as compared to the total size of all the local models learned by local classifier approaches. Also, in the global classifier approach, a single classification model is built from the training set, taking into account the label hierarchy and relationships. During the prediction phase, each test example is classified using the induced model, in a process that can assign labels to a test example at potentially every level of the hierarchy. Because of that, in this study we compare PCTs for MTP (as flat, global MLC approach) and PCTs for HMC (in a global setting) [3].

4 Experimental design

4.1 Datasets and evaluation measures

We use four multi-label classification benchmark problems used in previous studies and evaluations of methods for multi-label learning. Table 2 presents the basic statistics of the datasets. The datasets come from the domain of text categorization and pre-divided into training and testing parts as used by other researchers.

In any multi-label experiment, it is essential to include multiple and contrasting measures because of the additional degrees of freedom that the multi-label

Table 2. Description of the benchmark problems in terms of number of training (*#tr.e.*) and test (*#t.e.*) examples, number of features (D), total number of labels (Q) and label cardinality - average number of labels per example (l_c).

	Reference	*#tr.e.*	*#t.e.*	D	Q	l_c
tmc2007	[15]	21519	7077	500	22	2.16
bibtex	[16]	4880	2515	1836	159	2.40
bookmarks	[16]	60000	27856	2150	208	2.03
delicious	[2]	12920	3185	500	983	19.02

setting introduces. In our experiments, we used various evaluation measures that have been suggested by [11] In particular, we used 12 *bipartitions-based* evaluation measures: six *example-based* evaluation measures (*hamming loss*, *accuracy*, *precision*, *recall*, *F measure* and *subset accuracy*) and six *label-based* evaluation measures (*micro precision*, *micro recall*, *micro* F_1, *macro precision*, *macro recall* and *macro* F_1). Note that these evaluation measures require predictions stating that a given label is present or not (binary 1/0 predictions). However, most predictive models predict a numerical value for each label and the label is predicted as present if that numerical value exceeds some pre-defined threshold τ. The performance of the predictive model thus directly depends on the selection of an appropriate value of τ.

Also, we used four *ranking-based* evaluation measures (*one-error*, *coverage*, *ranking loss* and *average precision*) that compare the predicted ranking of the labels with the ground truth ranking. A detailed description of the evaluation measures can be found in [1].

4.2 Experimental setup

The comparison of the multi-label learning methods was performed using the CLUS[1] system for predictive clustering. All experiments were performed on a server with an Intel Xeon processor at 2.5GHz and 64GB of RAM with the Fedora 14 operating system. We used the default settings of CLUS to learn the single PCT approaches (PCTs for MTP - as flat MLC approach, and PCTs for HMC). The threshold τ for the *bipartitions-based* evaluation measures was set to 0.5 for all compared methods.

The balanced k-means clustering method requires to be configured the number of clusters k in each node of the hierarchy. For this parameter, five different values (2-6) were considered in the cross-validation phase [2]. After determining the best value of k on every dataset (via cross-validation on the training dataset), the PCT for HMC was trained using all available training examples and was evaluated by recognizing all test examples from the corresponding dataset. The values of the parameter k are 3 for the tmc2007, bibtex and bookmarks

[1] http://clus.sourceforge.net

datasets, and 4 for the delicious dataset. Also, for the balanced k-means and the agglomerative methods, Euclidean distance was used as a distance measure.

5 Results and discussion

In this section, we present the results from the experimental evaluation. Table 3 shows the predictive performance of the compared methods:

- PCTs for MTP, that don't use a hierarchy for solving the original MLC problem (labeled as *no hierarchy (flat MLC)*)
- PCTs for HMC, that use data-derived label hierarchies, defined by:
  - balanced k-means clustering approach (labeled as *balanced-k-means*)
  - agglomerative clustering by using complete linkage (labeled as *agglomerative (complete)*)
  - agglomerative clustering by using single linkage (labeled as *agglomerative (single)*)
  - predictive clustering trees (labeled as *PCTs*)

The first column of the table describes the methods used for defining the hierarchies, while the other columns show the predictive performance of the compared methods and hierarchies in terms of the 16 performance evaluation measures. The best results per dataset are shown in boldface.

Inspecting Table 3, we note that PCTs for HMC outperform PCTs for MLC on all datasets and on almost all evaluation measures. The instantiation of PCTs for MTP (for solving flat multi-label classification problems) shows better predictive performance only on *micro precision* evaluation measure on the *bibtex* and *bookmarks* datasets.

PCTs for HMC that use balanced k-means clustering for deriving the label hierarchies outperform PCTs for HMC that use agglomerative clustering with single and complete linkage and PCTs for deriving the label hierarchies on datasets with higher number of labels (*bibtex*, *bookmarks* and *delicious*). PCTs for HMC with agglomerative clustering perform the best on *tmc2007* dataset. The two agglomerative clustering methods (single and complete linkage) derived identical label hierarchies on all MLC problems, which result in same predictive performance in the experimental evaluation.

The highest improvement of utilizing the data-derived hierarchies is obtained on *delicious* dataset, as a result of the largest number of labels and the largest label cardinality (average number of labels per example). A large number of labels and large label cardinality yields a larger hierarchy that emphasizes the relations between labels, and improves the process of learning and prediction.

Table 3. The predictive performances of PCTs for MLC obtained on the original (flat) MLC problems and PCTs for HMC obtained on the transformed (newly) defined HMC problems by using four different clustering approaches (balanced k-means, predictive clustering trees, and agglomerative clustering with complete and single linkage) along 16 performance evaluation measures.

	HammingLoss	Accuracy	Precision	Recall	Fmeasure	SubsetAccuracy	MicroPrecision	MicroRecall	MicroF1	MacroPrecision	MacroRecall	MacroF1	OneError	Coverage	RankingLoss	AvgPrecision
tmc2007																
no hierarchy (flat MLC)	0.075	0.436	0.659	0.478	0.554	0.215	0.689	0.454	0.547	0.386	0.235	0.263	0.307	4.57	0.100	0.700
balanced-k-means - HMC	**0.067**	0.515	0.688	0.604	0.643	**0.253**	0.704	0.563	0.625	**0.735**	0.341	0.409	0.246	**3.35**	0.066	0.774
agglomerative (complete) - HMC	0.068	0.501	0.699	0.571	0.628	0.250	0.717	0.524	0.605	0.629	0.283	0.344	0.247	3.54	0.071	0.767
agglomerative (single) - HMC	0.068	0.501	0.699	0.571	0.628	0.250	0.717	0.524	0.605	0.629	0.283	0.344	0.247	3.54	0.071	0.767
PCTs - HMC	0.101	**0.559**	**0.746**	**0.703**	**0.723**	0.184	**0.742**	**0.625**	**0.678**	0.675	**0.358**	**0.418**	**0.084**	11.64	**0.055**	**0.835**
bibtex																
no hierarchy (flat MLC)	**0.014**	0.046	0.140	0.046	0.069	0.004	**1.000**	0.057	0.108	0.006	0.006	0.006	0.783	58.60	0.256	0.212
balanced-k-means - HMC	0.015	**0.243**	**0.368**	**0.290**	**0.324**	0.113	0.550	**0.259**	**0.352**	**0.296**	**0.174**	**0.202**	**0.449**	**30.36**	**0.105**	**0.491**
agglomerative (complete) - HMC	**0.014**	0.175	0.289	0.183	0.225	0.103	0.749	0.145	0.243	0.079	0.044	0.052	0.589	45.74	0.190	0.341
agglomerative (single) - HMC	**0.014**	0.175	0.289	0.183	0.225	0.103	0.749	0.145	0.243	0.079	0.044	0.052	0.589	45.74	0.190	0.341
PCTs - HMC	**0.014**	0.197	0.328	0.204	0.251	**0.117**	0.796	0.161	0.268	0.082	0.056	0.062	0.541	36.93	0.152	0.388
bookmarks																
no hierarchy (flat MLC)	**0.009**	0.133	0.133	0.137	0.135	0.129	**0.947**	0.076	0.141	0.018	0.016	0.017	0.817	73.78	0.258	0.213
balanced-k-means - HMC	**0.009**	**0.205**	**0.224**	**0.211**	**0.217**	**0.188**	0.776	**0.139**	**0.236**	**0.299**	**0.071**	**0.097**	**0.651**	**50.46**	**0.169**	**0.370**
agglomerative (complete) - HMC	**0.009**	0.160	0.163	0.165	0.164	0.153	0.875	0.097	0.175	0.103	0.026	0.030	0.729	57.99	0.200	0.302
agglomerative (single) - HMC	**0.009**	0.160	0.163	0.165	0.164	0.153	0.875	0.097	0.175	0.103	0.026	0.030	0.729	57.99	0.200	0.302
PCTs - HMC	**0.009**	0.177	0.185	0.181	0.183	0.167	0.846	0.110	0.195	0.116	0.036	0.044	0.699	56.31	0.193	0.328
delicious																
no hierarchy (flat MLC)	0.019	0.001	0.001	0.001	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.592	691.62	0.172	0.206
balanced-k-means - HMC	**0.018**	**0.118**	**0.429**	**0.132**	**0.201**	**0.007**	**0.621**	**0.120**	**0.201**	**0.162**	**0.049**	**0.062**	**0.386**	**548.01**	**0.121**	**0.336**
agglomerative (complete) - HMC	0.019	0.074	0.354	0.081	0.132	0.003	0.590	0.077	0.136	0.064	0.018	0.022	0.440	558.78	0.131	0.293
agglomerative (single) - HMC	0.019	0.074	0.354	0.081	0.132	0.003	0.590	0.077	0.136	0.064	0.018	0.022	0.440	558.78	0.131	0.293
PCTs - HMC	0.019	0.097	0.376	0.107	0.167	0.002	0.609	0.101	0.173	0.066	0.029	0.034	0.418	553.65	0.128	0.316

6 Conclusions and further work

In this paper, we have investigated the use of label hierarchies in multi-label classification, constructed in a data-driven manner. We consider flat label-sets and construct label hierarchies from the label sets that appear in the annotations of the training data by using clustering approaches based on balanced k-means clustering, agglomerative clustering with single and complete linkage, and clustering performed by PCTs. The hierarchies are then used in conjunction with hierarchical multi-label classification approaches in the hope of achieving better multi-label classification.

In particular, we investigate and evaluate the utility of four different data-derived label hierarchies in the context of predictive clustering trees for HMC. The experimental results clearly show that the use of the hierarchy results in improved performance and the more balanced hierarchy offers better representation of the label relationships.

The label hierarchies used in PCTs for HMC greatly improve the performance of PCTs for MTP (as used for MLC): The results show improvement in performance on almost all evaluation measures considered. Multi-branch hierarchy (defined by balanced k-means clustering) outperforms binary hierarchies (defined by agglomerative clustering with single and complete linkage and PCTs) on datasets with higher number of labels (*bibtex*, *bookmarks* and *delicious*). This improvement is especially emphasized on the *delicious* dataset, as a result of the higher label cardinality that this dataset has in comparison to the other evaluated datasets.

The final recommendation considering the performance of the evaluated methods is that we should use data-derived label hierarchies. We should transform the original (flat) multi-label classification problem into hierarchical multi-label one by using more balanced hierarchies, and solve the newly defined hierarchical classification problem by a classifier that can directly handle HMC problems.

We plan to extend this study by using more multi-label classification datasets, in particular more diverse ones. These would include different numbers of possible labels, different numbers of labels per example and different joint distribution properties for the labels (e.g., different degrees of (in)dependence among the labels). This would allow us to draw stronger conclusions on the conditions under which the use of a hierarchy on the label space and the way of its construction improves the performance of the different MLC approaches.

A final direction for further work might be the comparison of hierarchies constructed by humans and hierarchies generated in a data-driven fashion. For HMC problems, we can consider the MLC task defined by the leaves of the provided label hierarchy. We can then construct label hierarchies automatically, as described above, and compare these hierarchies (and their utility) to the originally provided label hierarchy.

Acknowledgements

We would like to acknowledge the support of the European Commission through the project MAESTRA - Learning from Massive, Incompletely annotated, and Structured Data (Grant number ICT-2013-612944).

References

1. Madjarov, G., Kocev, D., Gjorgjevikj, D., Dzeroski, S.: An extensive experimental comparison of methods for multi-label learning. Pattern Recognition **45**(9) (2012) 3084 – 3104
2. Tsoumakas, G., Katakis, I., Vlahavas, I.: Effective and Efficient Multilabel Classification in Domains with Large Number of Labels. In: Proc. of the ECML/PKDD Workshop on Mining Multidimensional Data. (2008) 30–44
3. Kocev, D.: Ensembles for predicting structured outputs. PhD thesis, IPS Jožef Stefan, Ljubljana, Slovenia (2011)
4. Tsoumakas, G., Katakis, I.: Multi Label Classification: An Overview. International Journal of Data Warehouse and Mining **3**(3) (2007) 1–13
5. Mencía, E.L., Park, S.H., Fürnkranz, J.: Efficient voting prediction for pairwise multilabel classification. Neurocomputing **73** (2010) 1164–1176
6. Blockeel, H., Raedt, L.D., Ramon, J.: Top-down induction of clustering trees. In: Proc. of the 15th International Conference on Machine Learning. (1998) 55–63
7. Vens, C., Struyf, J., Schietgat, L., Džeroski, S., Blockeel, H.: Decision trees for hierarchical multi-label classification. Machine Learning **73**(2) (2008) 185–214
8. Manning, C.D., Raghavan, P., Schütze, H.: An Introduction to Information Retrieval. Cambridge University Press (2009)
9. Kocev, D., Vens, C., Struyf, J., Džeroski, S.: Tree ensembles for predicting structured outputs. Pattern Recognition **46**(3) (2013) 817–833
10. de Carvalho, A.C.P.L.F., Freitas, A.A.: A tutorial on multi-label classification techniques. In Abraham, A., Hassanien, A.E., Snsel, V., eds.: Foundations of Computational Intelligence (5). Volume 205 of Studies in Computational Intelligence. Springer (2009) 177–195
11. Tsoumakas, G., Katakis, I., Vlahavas, I.: Mining multi-label data. In: Data Mining and Knowledge Discovery Handbook. Springer Berlin / Heidelberg (2010) 667–685
12. Silla, CarlosN., J., Freitas, A.: A survey of hierarchical classification across different application domains. Data Mining and Knowledge Discovery **22** (2011) 31–72
13. Dimitrovski, I., Kocev, D., Loskovska, S., Deroski, S.: Fast and scalable image retrieval using predictive clustering trees. In: Discovery Science. Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 33–48
14. Levatić, J., Kocev, D., Džeroski, S.: The use of the label hierarchy in HMC improves performance: a case study in predicting community structure in ecology. In: Proc. of the Workshop on New frontiers in mining complex patterns held in conjunction with ECML/PKDD2013. (2013) 189–201
15. Srivastava, A., Zane-Ulman, B.: Discovering recurring anomalies in text reports regarding complex space systems. In: Proc. of the IEEE Aerospace Conference. (2005) 55–63
16. Katakis, I., Tsoumakas, G., Vlahavas, I.: Multilabel Text Classification for Automated Tag Suggestion. In: Proc. of the ECML/PKDD Discovery Challenge. (2008)

A comparison of classification methods for gene prediction in metagenomics

Fabiana Goés[1], Ronnie Alves[1,2], Leandro Corrêa[1], Cristian Chaparro[2] and Lucinéia Thom[3]

[1] PPGCC - Federal University of Pará, Belém, Brazil
fabii.goes@gmail.com
[2] Vale Institute of Technology, Belém, Brazil
ronnie.alves@itv.org, cristian.chaparro@itv.org
[3] PPGC - Federal University of Rio Grande do Sul, Porto Alegre, Brazil
lucineia@inf.ufrgs.br

Abstract. Metagenomics is an emerging field in which the power of genome analysis is applied to entire communities of microbes. It is focused on the understanding of the mixture of genes (genomes) in a community as whole. The gene prediction task is a well-known problem in genomics, and it remains an interesting computational challenge in metagenomics too. A large variety of classifiers has been developed for gene prediction though there is lack of an empirical evaluation regarding the core machine learning techniques implemented in these tools. In this work we present an empirical performance comparison of different classification strategies for gene prediction in metagenomic data. This comparison takes into account distinct supervised learning strategies: one lazy learner, two eager-learners and one ensemble learner. The ensemble-based strategy has achieved the overall best result and it is competitive with the prediction baselines of well-known metagenomics tools.

Keywords: Machine learning, classification methods, gene prediction, metagenomics

1 Introduction

Since the human genome project several computation strategies have been developed to shed a light on the amazing complexity of the complex human genome. Completed in 2003, this international research effort provided, for the fist time, the blueprint for building a human being. Nowadays, we are facing a new voyage of discovery into the microorganism world. Microbial communities support all life on Earth, and metagenomics is a revolutionary new approach to better understanding the microbial world. This new science opens doors to a large amount of scientific exploration and can help understand some of the most complex medical, agricultural, environmental, and economic challenges of today's world [1, 2].

Metagenomics is the application of shotgun sequencing to DNA obtained directly from an environmental sample or series of related samples, and it is also a derivation of conventional microbial genomics, with the key difference being that it bypasses the requirement for obtaining pure cultures for sequencing [3]. It is focused on the understanding of the mixture of genes (genomes) in a community as a whole [4]. The gene prediction task is a well-known problem in genomics, and it remains an interesting computational challenge in metagenomics too.

Gene prediction is the procedure of finding protein and RNA coding sequences in the sample DNA. Depending on the applicability and success of the assembly, gene prediction can be done on post assembly *contigs*[1], on reads from unassembled metagenomes or on a mixture of *contigs* and individual unassembled reads. There are two main strategies for gene prediction [3]: i) **evidence-based** gene-calling methods use homology searches to find genes similar to those observed previously (reference microbial genomes); and ii) **ab initio** gene-calling relies on the intrinsic features of the DNA sequence to discriminate between coding and noncoding regions, allowing for the identification of homologs in the available databases. The former approach has two major drawbacks. Low values of similarity to known sequences either due to evolutionary distance or due to the short length of metagenomic coding sequences and the presence of sequence errors restrict the identification of homologs. In addition, novel genes without similarities are completely ignored. The latter approach usually employs Machine Learning (ML) algorithms which can smooth the previous gene prediction drawbacks. Still this requires a proper use of sophisticated classification methods and careful selection of potential DNA sequence features that could best discriminate between coding and noncoding sequences.

A large variety of classifiers has been developed for gene prediction. The hidden Markov models (HMM) is the state-of-the-art ML technique used since the 90's, and it is at the core of the pipeline called *GeneMark.hmm* [5]. Recently, metagenomic pipelines have adopted new classification strategies such as i) support vector machines(SVM) [6] (*MetaGUN*) and ii) artificial neural networks (ANN) [7](*Orphelia*). As an example, in *Orphelia* first a linear discrimination analysis takes place to select candidate features followed by ANN that calculates the probability of an ORF[2] being a potential coding sequence.

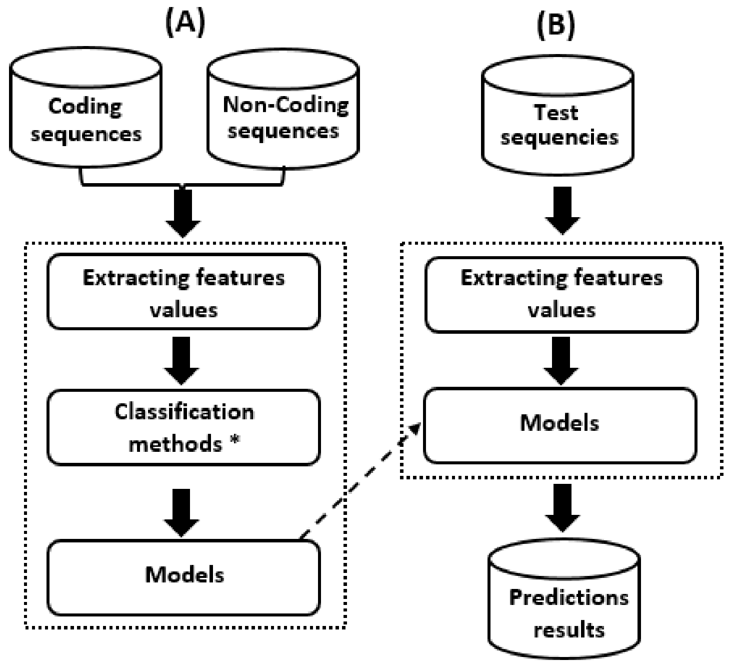
Applications of supervised machine learning methodologies continue to grow in the scientific literature across several domains. Jensen and Bateman conducted a careful text-mining over several biomedical articles in PubMed and they observed a moderate decrease in the use of both ANN and HMM, and an increase in usage of SVM and Random Forest in the literature [8]. This study takes into account, basically, the citation of these ML methods. In this work we present an empirical performance evaluation of the core ML techniques explored for gene prediction by some of the most used metagenomic pipelines.

[1] A contig is a continuous sequence resulting from the assembly of overlapping small DNA fragments (sequence reads).

[2] An ORF is a sequence of DNA that starts with a start codon, usually "ATG", and ends with any of the three termination codons (TAA, TAG, TGA).

2 Materials and Methods

In Figure 1 we depict the overall architecture devised for the comparison of the classifiers. It follows the classical steps of data preprocessing, learning and test. First, coding and non-coding sequences are extracted for the identification of potential sequence features, and next classification models are built for further prediction analysis (Figure 1-A). Once new sequences are retrieved it is possible to classify them in accordance with the classification models, and thus, an appreciation regarding whether it is a coding sequence or not can be done(Figure 1-B).



*Classification Methods: Random Forest, K-Nearest-Neighbor, Neural Network and Support Vector Machines.

Fig. 1: The overall architecture devised for the comparison of the classification methods.

2.1 Classification methods

We have selected four classification strategies for the comparison study. These methods employ distinct learning strategies, and ideally, each one has a particular manner to generalize the search space. The gene prediction problem is simply a binary classification or *concept learning* (positive class: coding sequence and negative class: no coding sequence). This comparison takes into account distinct supervised learning strategies: one lazy learner (KNN: K-Nearest Neighbors), two eager-learner (SVM: Support Vector Machines and ANN: Artificial Neural Networks) and one ensemble learner (RF: Random Forest).

Random forest (RF) It is a well-known ensemble approach for classification tasks proposed by Breiman [9]. Its basis comes from the combination of tree-structured classifiers with the randomness and robustness provided by bagging and random feature selection. Several decision trees are trained with random bootstrap samples from the original data set (2/3 of data) and afterwards, results are combined into a single prediction: for classification tasks, by means of voting; for regression tasks, by averaging the results of all trees. The fact that the predicted class represents the mode of all the classes output by individual trees gives robustness to this ensemble classifier in relation to a single tree classifier. Given its basis on an ensemble learning it is less impacted by $overfitting$, making it a potential candidate ML approach in bioinformatics problems [10]. Though, as far as we know it has not been explored as a solution in the gene prediction of metagenomic sequences.

K-Nearest Neighbors (KNN) Nearest-neighbor classifiers are based on learning by analogy, by comparing a given test instance with training instances that are similar to it [11]. The training instances are described by n features. Each instance represents a point in a n-dimensional space. Thus, all of the training instances are stored in an n-dimensional pattern space. When an unknown instance is provided, a k-nearest-neighbor classifier searches the pattern space for the k training instances closest to the unknown instance. *Closeness* is usually defined in terms of a distance metric, such as Euclidean distance. K-nearest neighbors are also used as a baseline strategy for comparison among distinct classifiers.

Artificial Neural Networks (ANN) A neural network is a set of connected input/output units in which each connection has a weight associated with it. During the learning stage, the network learns by adjusting the weights with aims to predict the correct class label of the input instances. ANN also involves long training times and are also criticized for their poor interpretability. Nevertheless, it has a higher tolerance to noisy data as well as the ability to classify patterns on which it has not been trained. *Backpropagation* is the most popular ANN algorithm and it performs learning on a *multilayer feed-forward* neural network [11]. A *multilayer feed-forward* neural network basically consists of an input layer, one or more hidden layers, and an output layer.

Support Vector Machines (SVM) It uses a linear model to implement nonlinear class boundaries. SVM transform the input using a nonlinear mapping, thus, turning the instance space into a new space. A linear model (the *maximum* *margin hyperplane*) constructed in the new space can represent a nonlinear decision boundary in the original space. The *maximum margin hyperplane* is the one that gives the greatest separation between classes. The instances that are closest to this hyperplane, so the ones with minimum distance to it, are called *support vectors*. Other kernel functions can be used instead to implement distinct nonlinear mappings. Two that are often suggested are the *radial basis functions (RBF) kernel* and the *sigmoid kernel*. These functions do not present large differences in terms of prediction accuracy, though this observation depends on the application domain. SVM has been used extensively in several domains, and in some cases it outperforms ANN [12].

2.2 Feature engineering

Feature engineering is at the core of classification strategies and it is a crucial step on prediction modeling. Essentially, two different types of information are currently used to try to find genes in a genomic sequence: i) *extrinsinc content sensors* explore a sufficient similarity between a genomic sequence region and a protein or DNA sequence present in a database in order to determine whether the region is transcribed and/or coding; and ii) *intrinsic content sensors* proposed particularly for prokaryotic genomes, in which features that characterize the sequence as "coding" for a protein are carefully calculated for discrimination analysis [13, 14]. Examples of content sensors are: nucleotide composition and especially $(G + C)$ content (introns being more A/T-rich than exons, especially in plants), codon composition, hexamer frequency, base occurrence periodicity, etc. Hexamer usage has been widely exploited by a large number of algorithms through different methods [14]. Table 1 presents six content sensors that are strongly used by gene prediction tools in metagenomics. For the comparison study we focused on three main types of features: $(G + C)$ content, length and codon usage. From this information we derived a total of six features as follows: 1) GC content, 2) GC content in the first position of each codon, 3) GC content in the second position of each codon, 4) GC content in the third position of each codon, 5) the sequence length, 6) the codon usage variance among the 61 monocodons.

	GC Content	Length	Codon usage	Dicodon usage	TIS	Aminoacid usage
Orphelia	x	x	x	x	x	
MetaGUN		x	x		x	
MGC	x	x	x	x	x	x
MetaGene	x		x	x		
FragGeneScan			x			

Table 1. Content sensors features used [x] by gene prediction tools in metagenomics.

GC-content It is the percentage of guanine and cytosine bases in all bases of a sequence. It has been used extensively by several gene prediction tools. This utilization is mainly due to the fact that coding regions present, on average, a higher GC content than on non coding sequences [15]. Differently from previous studies (see Table 1), we calculated the total level of GC content, and the content at the first, second and third monocodon positions with the aim to evaluate their impact in the gene prediction task. In this way, four features are derived from the GC content.

Length Another feature for discrimination between coding and non-coding sequence is its length. The intergenic regions are usually smaller than coding regions[14].

Codon Usage Perhaps the most important features for the discrimination between coding and non-coding sequences can be calculated from codon usage [16], in particular the frequencies of 4^3 monocodons. These frequencies represent the occurrences of successive trinucleotides (non-overlapping). For the characterization of monocodon usage, we compute the variance among the 61 monocodons, since gene sequences do not contain stop codons.

2.3 Training Data

The training data is basically DNA sequences having both coding sequences (positive) and intergenic regions (negative) instances. Our approach to compare the four classification methods is based on a learning scheme over eight prokaryotic genomes, namely two *Archaeas* and six *Bacterias*, available in GenBank[3] (Table 2). The choice of these organisms has to do with the experimental metagenomic data that will be evaluated while testing the predictive models. Thus, either these organisms belong to the same branch of the evolutionary tree or they are associated to Acid Mine Drainage biofilms (Section 2.4).

We have developed an algorithm to properly extract the coding and non-coding regions, on both forward and reverse strands, from these eight "complete" genomes. This algorithm was applied to regions with sequence lengths higher than 59 bp. Sequences less than 60 bp are ignored since they are too short to provide useful information [6]. Those originating from the annotated genes are used as positive instances of coding sequences, whereas others are treated as items of the non-coding class. After running this procedure we came up with a total of 30144 sequences, being 10106 related to intergenic regions and remaining 20038 of coding sequences.

2.4 Test Data

The metagenomic data selected for the comparison study is the Acid Mine Drainage (AMD) biofilm [17], freely available at the site of NCBI [4]. This biofilm

[3] http://www.ncbi.nlm.nih.gov/news/10-22-2013-genbank-release198
[4] http://www.ncbi.nlm.nih.gov/books/NBK6860/

Species	GenBank Acc.
Thermoplasma acidophilum *	NC_002578
Thermoplasma volcanium *	NC_002689
Acidimicrobium ferrooxidans	NC_013124
Acidithiobacillus caldus	NC_015850
Acidithiobacillus ferrooxidans	NC_011206
Acidithiobacillus ferrivorans	NC_015942
Candidatus Nitrospira defluvii	NC_014355
Thermodesulfovibrio yellowstonii	NC_011296

Table 2. The prokaryotic genomes used as reference for the training data. The "*" symbol highlights the two *Archaeas*.

sequencing project was designed to explore the distribution and diversity of metabolic pathways in acidophilic biofilms. Acidophilic biofilms are self-sustaining communities that grow in the deep subsurface and receive no significant inputs of fixed carbon or nitrogen from external sources. While some AMD is caused by the oxidization of rocks rich in sulfide minerals, this is a very slow process and most AMD is due directly to microbial activity [18]. More information regarding the AMD study as well as environmental sequences, metadata and analysis can be obtained at [17].

We have selected prokaryotic genomes associated to the same species found in Tyson[17]. Thus, five genomes (2 *Archaeas* and 3 *Bacterias*) were extracted from GenBank to create the test data (Table 3).

Species	GenBank Acc.
FA: Ferroplasma acidarmanus *	NC_021592
TA: Thermoplasmatales archaeon BRNA *	NC_020892
LFI: Leptospirillum ferriphilum	NC_018649
LFO: Leptospirillum ferrooxidans	NC_017094
SA: Sulfobacillus acidophilus	NC_015757

Table 3. The prokaryotic genomes used as reference for the test data. The "*" symbol highlight the two *Archaeas*.

2.5 Measures of prediction performance

The classifiers will be evaluated through the evaluation of classical prediction performance measures, namely, accuracy (ACC), specificity (SPE), sensitivity (SEN) and Kappa. All these measures are easily calculated from the resulting confusion matrix for each classifier. This matrix usually has two rows and two columns that reports the number of false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN). Though we provide the results for all these measures, we believe that Kappa is the most suitable measure to compare distinct classifiers. Kappa measures how closely the instances labeled by the classifiers matched the data labeled as *ground truth*, controling for the

ACC of a random classifier as measured by the expected accuracy. Thus, the kappa for one classifier is properly comparable to others kappa's classifiers for the same classification task.

$$ACC = \frac{TP+TN}{P+N} \quad (1)$$

$$SPE = \frac{TN}{TN+FP} \quad (2)$$

$$SEN = \frac{TP}{TP+FN} \quad (3)$$

$$Kappa = \frac{Pr(a)-Pr(e)}{1-Pr(e)} \quad (4)$$

3 Results and Discussion

3.1 Performance of the classifiers

The prediction modeling and evaluation was carried out with the *caret R* package [19]. We use the built-in *tune*() function for resampling and tuning to optimize all classifiers parameters. The best values were as follows: i) RF (mtry 4), KNN (k=5), ANN (size=5 and decay=0.1), SVML (C=0.5). The performance measures were calculated from the average performance of three repetition of a 10-fold cross validation scheme (Table 4). The most promising results were obtained by the RF model using 200 trees (Figure 2).

	ACC	KAPPA
RF model	**0.94**	**0.87**
KNN model	0.87	0.70
ANN model	0.91	0.80
SVML model	0.88	0.74

Table 4. The average performance of the classifiers. The orange cells highlight the best performance achieved by the RF classifier.

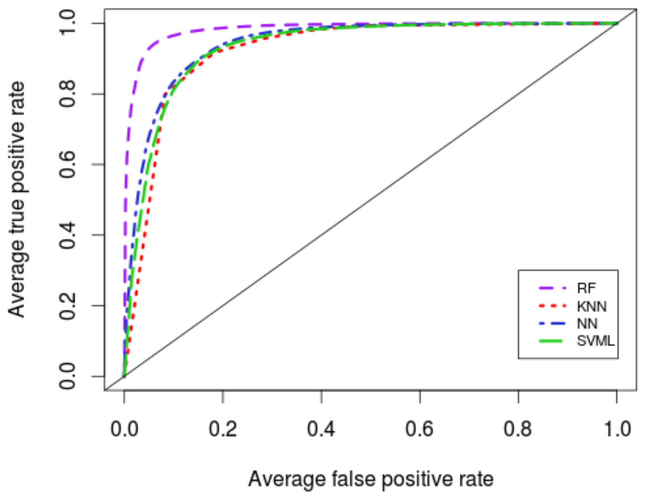


Fig. 2: RF has the best performance among all classifiers.

3.2 Comparison of classifiers using independent test data

In this section we present the performance comparison of the selected classifiers using the independent test data discussed in Section 2.4. So, the main goal was to evaluate how classifiers correctly classify the known coding sequences for the species associated to the AMD metagenome. As we expected the ensemble learning classifier employed by RF has achieved the best performance among all classifiers (Table 5 and Table 6). Ensembles are designed to increase the accuracy of a single classifier by training several distinct classifiers and combine their decisions to output a single class label. Given such accuracy-oriented design, ensemble learning algorithms are less likely to $overfitting$ when dealing with imbalanced data.

The SVM has an overall performance similar to KNN (base classifier), and this is partially due to the generalization carried by a linear SVM. Probably a radial SVM model would be able to generalize better the search space. On the other hand, the other eager learner, ANN, presents competitive results. As an example, ANN outperforms RF for the LFI specie (Kappa=0.9097).

Let us assume that we built a gene prediction method that is solely based on a RF model, so the overall performance of our model would have SEN=0.91 for a "hypothetical" metagenome (as the one discussed in Section 2.4). Table 7 presents a prediction baseline discussed in [6], where the $MetaGUN$ tool, based on a SVM classifier, outperforms the other two well-known gene prediction *pipelines*. Though SVM would hypothetically outperform our RF model, it is important to mention that our feature set is less complex than the one employed by *MetaGUN*.

	SEN				SPE			
Species	RF	ANN	KNN	SVML	RF	ANN	KNN	SVML
FA	**0.9380**	0.8801	0.6337	0.6521	0.9047	0.8642	0.9503	0.8662
LFI	**0.8945**	0.8783	0.8139	0.8203	0.9304	0.9316	0.9352	0.9276
LFO	**0.8797**	0.8469	0.7939	0.7743	0.9599	0.9628	0.9570	0.9504
SA	**0.9317**	0.9017	0.8145	0.8517	0.9433	0.9398	0.9486	0.9267
TA	**0.9085**	0.8408	0.7711	0.7642	0.9889	0.9679	0.9640	0.9522

Table 5. The comparison performance of classifiers in accordance to the SEN and SPE measures. The highlighted cells show the best results.

	ACC				Kappa			
Species	RF	ANN	KNN	SVML	RF	ANN	KNN	SVML
FA	0.9173	0.8702	0.8302	0.785	**0.8275**	0.7298	0.6182	0.5317
LFI	0.9156	0.9097	0.8854	0.8835	**0.8256**	**0.9097**	0.7599	0.7565
LFO	0.9263	0.9143	0.8888	0.8767	**0.8472**	0.8213	0.7666	0.741
SA	0.9383	0.9235	0.8913	0.8947	**0.8741**	0.8434	0.7746	0.7834
TA	0.957	0.9175	0.8875	0.9175	**0.9089**	0.8243	0.7577	0.737

Table 6. The comparison performance of classifiers in accordance to the ACC and Kappa measures. The highlighted cells show the best results.

	1200 bp		870 bp		535 bp		120 bp	
	Sen	Spec	Sen	Spec	Sen	Spec	Sen	Spec
MetaGUN (SVM)	**97.7**	94,8	97.4	95.2	96.9	95.4	93.2	89.6
FragGeneScan (Markov Models)	**95.7**	87.3	95.5	88.0	95.2	88.4	90.4	82.1
Orphelia (Neural Network)	**94.6**	94.7	94.1	94.7	93.3	94.6	82.0	76.4

Table 7. The performance of three well-known pipelines for gene prediction in metagenomic data. The highlighted cells show the best results obtained for detecting the real signal (gene).

4 Conclusions

Gene prediction is a well-known computational challenge in both genome and metagenome analysis. This latter poses an even more difficult problem since: i) metagenomes are a mixture of several distinct genomes and ii) most of the available genomes are not completed, so mainly *draft* genomes are available. Therefore, the task of gene prediction is *biased* in the proper selection of potential features in this complex domain as well as the choice of a robust machine learning algorithm.

In this work we presented an empirical comparison of several well-known classification methods applied to gene discovery in experimental metagenomic data. Though the performance of the four base classifiers was good, the ensemble-based strategy *Random Forest* has achieved the overall best result. As it can be observed by the associated ML literature. Ensemble learning strategies such as Random forest has been successfully applied on a large variety of business and scientific applications. Basically such observation is due to the fact the combination of models could be able to best generalize the hypothesis space. Though the best approach to combine models continues an open problem in ensemble learning research.

We plan to develop a new gene prediction *pipeline* having its basis on Random Forest. To the extent of our knowledge there is no reference of gene prediction algorithms based on a RF classifier.

Author's contributions

FG and RA performed the analysis and developped the pipeline. RA and CC supervised the study. FG, RA, CC and LT wrote the manuscript.

Acknowledgements

This work is partially supported by the Brazilian National Research Council (CNPq – *Universal calls*) under the BIOFLOWS project [475620/2012-7]. FG has a master scholarship by the Federal Agency for Support and Evaluation of Graduate Education (Capes).

References

1. Handelsman, J., Tiedje, J., Alvarez-Cohen, L., Ashburner, M., Cann, I., Delong, E., Doolittle, W., Fraser-Liggett, C., Godzik, A., Gordon, J., et al.: The new science of metagenomics: Revealing the secrets of our microbial planet. Nat Res Council Report **13** (2007)
2. Thomas, T., Gilbert, J., Meyer, F.: Metagenomics-a guide from sampling to data analysis. Microb Inform Exp **2**(3) (2012)
3. Kunin, V., Copeland, A., Lapidus, A., Mavromatis, K., Hugenholtz, P.: A bioinformatician's guide to metagenomics. Microbiology and Molecular Biology Reviews **72**(4) (2008) 557–578
4. Wooley, J.C., Godzik, A., Friedberg, I.: A primer on metagenomics. PLoS computational biology **6**(2) (2010) e1000667
5. Lukashin, A.V., Borodovsky, M.: Genemark. hmm: new solutions for gene finding. Nucleic acids research **26**(4) (1998) 1107–1115
6. Liu, Y., Guo, J., Hu, G., Zhu, H.: Gene prediction in metagenomic fragments based on the svm algorithm. BMC bioinformatics **14**(Suppl 5) (2013) S12

7. Hoff, K.J., Lingner, T., Meinicke, P., Tech, M.: Orphelia: predicting genes in metagenomic sequencing reads. Nucleic acids research **37**(suppl 2) (2009) W101–W105
8. Jensen, L.J., Bateman, A.: The rise and fall of supervised machine learning techniques. Bioinformatics **27**(24) (2011) 3331–3332
9. Breiman, L.: Random forests. Machine Learning **45**(1) (2001) 5–32
10. Strobl, C., Boulesteix, A.L., Kneib, T., Augustin, T., Zeileis, A.: Conditional variable importance for random forests. BMC bioinformatics **9**(1) (2008) 307
11. Han, J., Kamber, M., Pei, J.: Data mining: concepts and techniques. Morgan kaufmann (2012)
12. Faceli, K.: Inteligência artificial: uma abordagem de aprendizado de máquina. Grupo Gen-LTC (2011)
13. Ermolaeva, M.D., White, O., Salzberg, S.L.: Prediction of operons in microbial genomes. Nucleic acids research **29**(5) (2001) 1216–1221
14. Mathé, C., Sagot, M.F., Schiex, T., Rouzé, P.: Current methods of gene prediction, their strengths and weaknesses. Nucleic acids research **30**(19) (2002) 4103–4117
15. Fickett, J.W.: Recognition of protein coding regions in dna sequences. Nucleic acids research **10**(17) (1982) 5303–5318
16. Hoff, K.J., Tech, M., Lingner, T., Daniel, R., Morgenstern, B., Meinicke, P.: Gene prediction in metagenomic fragments: a large scale machine learning approach. BMC bioinformatics **9**(1) (2008) 217
17. Tyson, G.W., Chapman, J., Hugenholtz, P., Allen, E.E., Ram, R.J., Richardson, P.M., Solovyev, V.V., Rubin, E.M., Rokhsar, D.S., Banfield, J.F.: Community structure and metabolism through reconstruction of microbial genomes from the environment. Nature **428**(6978) (2004) 37–43
18. Johnson, D.B., Hallberg, K.B.: Acid mine drainage remediation options: a review. Science of the total environment **338**(1) (2005) 3–14
19. Kuhn, M.: The caret package homepage. URL http://caret. r-forge. r-project. org (2010)

Discovering Behavioural Patterns in Knowledge-Intensive Collaborative Processes

Claudia Diamantini, Laura Genga, Domenico Potena, and Emanuele Storti

Dipartimento di Ingegneria dell'Informazione
Università Politecnica delle Marche
via Brecce Bianche, 60131 Ancona, Italy
{c.diamantini,l.genga,d.potena,e.storti}@univpm.it

Abstract. Domains like emergency management, health care, or research and innovation development, are characterized by the execution of so-called *knowledge-intensive* processes, which require skilled personnel capable of facing complex issues which necessitate both judgment and creativity. Such processes are typically highly uncertain, with little or no structure; consequently, classical process discovery techniques, aimed at extracting complete process schemas from execution logs, usually obtain rather poor performances when applied on these processes. As a remedy, in the present work we propose a methodology aimed at extracting relevant subprocesses, representing meaningful collaboration behavioural patterns. Furthermore, we deal with the problem of the lack of a well-formed event log, which is typical in most creative and unstructured domains whose activities are not supported by dedicated information systems. We consider a real case study regarding the development of research activities, to test the approach and compare its results with the outcome of classical process discovery techniques.

Keywords: behavioural patterns discovery; knowledge-intensive processes; hierarchical clustering

1 Introduction

Nowadays process analysis techniques are more and more focused on the analysis of so-called *knowledge-intensive processes*, that are defined as processes whose value "can only be created through the fulfillment of knowledge requirements of the process participants" [9]. In other words, these processes usually require high-qualified and skilled personnel, capable of facing complex, ambiguous issues which necessitate both judgment and creativity [5]. Examples of knowledge-intensive processes can be found in emergency management, diagnosis and treatment in the health care domain, Research & Development, enterprise strategic planning, innovation development and so on. Such processes usually require to combine knowledge and competencies of different domains experts; indeed, they are typically managed by inter-disciplinary teams, whose members can also be physically distributed. As a result, an efficient management of collaboration is required in order to achieve the requested goals.

While typical business processes are usually driven by well-defined schemas, activities in knowledge-intensive processes are non-repetitive and difficult to plan; indeed, the actual activity flow is mainly established by the decisions of process participants, which usually depend on the particular context of process execution, thus introducing a high degree of variability. The author in [1] calls this kind of unstructured processes *spaghetti processes*, to distinguish them from structured ones, named *lasagna processes*. It's noteworthy that classical process discovery techniques, aimed at deriving complete process schemas from corresponding event logs, usually obtain poor results when applied to complex spaghetti processes. Although a number of alternative techniques were developed in the literature to deal with spaghetti processes (e.g. [10]), they mostly try to simplify the final outcome removing infrequent activities, hence loosing significant knowledge about the process. In the present work, we propose an alternative methodology for the analysis of spaghetti processes. Our approach moves from the extraction of the complete process schema towards the discovery of most relevant *behavioural patterns*, that is common work practices of teams involved in knowledge-intensive processes. These behavioural patterns provide a valuable support in the management of activities and for process improvement; for instance, they can be used to analyse the correlation between behaviours and desired/undesired process outcomes, or to identify bottlenecks and improve parallelism when possible. In particular, we intend to exploit a hierarchical graph clustering technique, which returns a taxonomy of behavioural patterns where patterns at the top level of the hierarchy are the most common ones and lower-level patterns are defined on the basis of upper-level ones.

Another relevant issue to face when dealing with knowledge-intensive processes, regards event logs. While in some domains, like health care, information systems exist supporting the activities tracking and, hence, the building of well-formed logs, this is not the case in more creative and unstructured domains, like research or innovation. In fact, involved team members usually perform their activities on an arbitrary number of digital tools, like emails, chats, collaborative document editing tools, shared calendars, social media and so forth. As a consequence, information about activities is spread in a number of logs of different formats, thus requiring significant data fusion efforts in order to obtain a single process event log. Our methodology takes into account the presence of multiple, heterogeneous data sources, thus resulting suitable also for such contexts where activity monitoring systems are not available.

The rest of this work is organised as follows. In Subsection 1.1 some related works are introduced; in Section 2 we introduce a real case study, that we will use through the paper; in Section 3 we explain the main phases of our methodology; Section 4 discusses some experiments we performed to validate our approach. Finally, in Section 5 we draw some conclusions, and delineate future works.

1.1 Related Work

Process Mining (PM) is a set of methodologies used to analyse process event logs, like the ones produced by ERP systems, Workflow Management Systems or other process-aware enterprise systems, to extract corresponding process schemas. Although some examples of usage of PM to analyse spaghetti processes exist, like [15, 11], regarding the software development domain, these techniques are typically applied to structured business processes, for which it is usually possible to derive a proper schema. However, when dealing with complex domains and unstructured processes, the adoption of a single schema to model such processes can likely originate too complex models or, on the contrary, oversimplified models, not so useful for the analyst.

Therefore, our approach is oriented to discover patterns instead of schemas, focusing on parts of the process (i.e. patterns) rather than on the whole process. In particular, the methodology we propose is based on the one introduced in [7], where a graph-based hierarchical clustering algorithm is used to discover patterns from process schemas. Clustering techniques previously proposed in the literature are mainly aimed at enhancing the quality of discovered process schemas [8, 16, 3], while the application of clustering techniques to process schemas themselves, as proposed in this paper, is almost new. To the best of our knowledge, the only similar approach is in [13]. Major differences between [13] and the proposal discussed here are: (1) the process schema is translated in vector format and then traditional agglomerative clustering techniques are used instead of exploiting graph clustering, and (2) clusters of whole processes are generated while similar substructures cannot be recognized.

As regards analysis of collaborative activities, some similarities with our work can be found in the *Computer Supported Collaborative Learning* (CSCL) field [14], aimed to grasp knowledge about the collaborative learning process. Our approach, however, differs from typical CSCL ones, which usually exploit a centralized and dedicated platform where all activities are carried out, thus allowing the storage of all needed data in a simple and efficient way. Examples of such an approach are *DIAS* [4] and *DEGREE* [2]. Also, in these works they produce general process indicators instead of extracting behavioural patterns.

2 Case Study: Collaborative Research Activity

To illustrate our methodology we introduce a real case study describing the development of a scientific paper performed by the authors, which represents a typical collaborative situation. More precisely, we consider a team of 4 people who have collaborated for approximately six months to develop the paper, i.e. from December 2012 to May 2013. During such a period, team members performed several different activities, among which we selected three main kinds of tasks, namely a) writing of the paper, which involved file editing, b) experimental activities, which regarded programming and, finally, c) communication and coordination activities, in particular emails and Skype discussions. We collected the logs from those tools with which each member of the team usually works. In particular, Dropbox for tasks of kind a), SVN for tasks b) and finally emails and Skype conversations for tasks c). Such a scenario presents some interesting issues to deal with. Firstly, collaboration activities are spread in the temporal dimension, since writing the paper required several months, where we observed relevant changes in team workload, usually more significant in certain periods (e.g. the days immediately before the submission deadline). Secondly, as usually happens, during the given period the whole research team (or just a subset) took part in several projects involving some external collaborators. Hence, to analyse a specific project we needed to isolate its activities. Moreover, we also had to filter activities related to private life of each member, usually stored in their PCs (e.g. chat with her family). Finally, the heterogeneous and distributed nature of data. Although we chose a limited number of software as data sources, team members had been using different versions of these tools running over different operating systems. In general, when dealing with collaborative tasks we have to consider an arbitrary large set of data sources, typically spread among team members machines and stored in an arbitrary way. Such kind of issues are quite common in most collaborative scenarios, thus allowing us to generalise the obtained results for several kinds of collaboration analysis problems. We introduce such a methodology in the next Section.

3 Methodology

Our methodology begins with the generation of log traces, which involves the collection of data from heterogeneous data sources and their transformation and integration into a single data log. Then, process traces are transformed into graphs in order to extract common and frequent sub-graphs by means of a graph mining clustering technique.

The log building and the event clustering phases are described in Subsection 3.1, whereas graph mining clustering techniques are described in Subsection 3.2.

3.1 Log building

In this phase we collect and manipulate raw data from several sources to obtain a single log. More precisely, first we have to *extract* data, by identifying interesting data sources; to this end, we need to involve team members, to know which tools they commonly use for their activities. In our example, we take into account Dropbox, SVN repository, emails and Skype.

Once we obtain the various logs, we *transform* them in the format we use for our analysis, where each event is described by its id, its timestamp, one or more actors (i.e. the "resource") and the process instance it refers to (i.e. the "case id"). During this transformation we apply some rules aimed at filtering noise events; for instance, as regards Dropbox events, we only consider events about files or folders whose names contain at least one of the keywords regarding the domain of the case study, e.g., the name and the acronym of the conference, the title of the paper, and so forth. It's noteworthy that while for Dropbox and SVN tools the transformation is quite straight, since they both produce logs in a format similar to the one we need, messaging tools require more efforts, since they do not provide any support for event tracking. As a remedy, we introduce the notion of "event", modeling it as a communicating act. More precisely, in asynchronous messaging tools an event consists in sending an email with a certain subject to someone, while in synchronous messaging tools the event consists in sending the first message of a chat. We consider as event resource the member who sends the email, in the first case, and all members which actively participate to the chat (i.e. they wrote at least one message), in the second one.

Finally, we have to *integrate* all converted logs into a single one. More precisely, firstly we have to integrate sources for each single member and, then, proceed with the final integration between all members logs. We can note, however, that the first step is mostly addressed in the previous phase, from which we obtain logs homogeneous with respect to the established format, that can be merged simply by ordering events on the basis of timestamps. Instead, the integration between logs of different team members requires to take into account both low-level issues, mainly regarding system and hardware heterogeneities (e.g. compare timestamps of not synchronized systems) and high-level ones, concerning team members working habits (e.g. different email contacts aliases). Strategies to deal with such heterogeneities mainly depend on the particular context. In our case, an example consists in the collection from each member of the list of email aliases of her contacts, in order to identify the people they refer to. Details on the whole procedure for log building are available in [6].

3.2 Hierarchical clustering

Once log traces have been obtained, we proceed by converting them in graphs. Such a conversion is quite straight: a node is created for each event in the trace, while each pair of subsequent events is linked through an edge. Once we obtained the graphs, it is possible to analyse them by exploiting a hierarchical clustering technique. Such techniques are aimed at extracting frequent substructures (i.e., sub-graphs) out of a set of input graphs. Such clusters are then arranged in a hierarchy of clusters, where the top-level clusters are defined only through elements belonging to input graphs (i.e., nodes and edges), while lower-level clusters extend upper-level clusters with other elements, implicitly defining a lattice structure. Therefore, descending the hierarchy, we pass from structures that are very common in input graphs (i.e., frequently occurring, with a high support) to structures specific for each input graph (i.e., with low support).

An example is shown in Figure 1 where a lattice is generated by a repository of graphs. A, B, C are specific activities and S_i are substructures. Graphs that contain the substructure S_1=$\{A \rightarrow A\}$ belong to cluster C_1, while graphs containing the substructure S_2=$\{B \rightarrow A;\ B \rightarrow C\}$ belong to C_2. The cluster C_3 is described by S_1 plus an additional node B. Therefore, C_3 is a specialization of C_1, because the former extends the latter, given that it contains the structure $\{A \rightarrow A \rightarrow B\}$. Finally, C_4 is child of both C_1 and C_2, since it is the set of graphs where S_1 is linked to S_2. Note that different clusters can be children of the same parents: for instance, a cluster described by the substructure $\{S_1\rightarrow A\rightarrow S_2\}$ would be sibling of C_4.

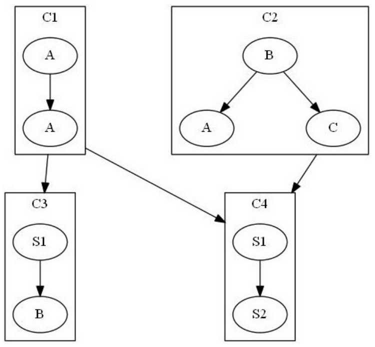


Fig. 1: A hierarchical clustering lattice

Among the hierarchical clustering algorithms, in this work we refer to Subdue [12] that, by iteratively analysing input graphs, is capable to extract at each step all existing substructures and to discover the one that best compresses the graphs. After each iteration, such a substructure is then actually used to compress the graphs, by replacing each occurrence of the substructure with a single node. Hence, the chosen substructure becomes a cluster of the lattice, and the compressed graphs are presented to Subdue again, in order to repeat these steps until no more compression is possible. The search for the best substructure is driven by the minimum DL (Description Length) criterion, aimed to the minimization of the description length of the graph after the compression, i.e., the number of bits needed to represent its adjacency matrix. In more details, the algorithm at each step tries to maximize the multiplicative inverse of the following compression index, that globally takes into account both the dimension of a substructure and its frequency, and is computed as $\frac{DL(S)+DL(G|S)}{DL(G)}$, where $DL(G)$ is the DL of the input graph, $DL(S)$ is the DL of the substructure S and $DL(G|S)$ is the DL of G compressed by S.

Despite the fact that the lattice is a clustering model, we can not use well-known clustering evaluation measures, namely intra- and inter-clusters measures: indeed, due to its hierarchical nature, a lattice is characterized by a high overlap among all children of the same cluster, thus making not suitable the typical evaluation measures. Hence, we have to refer to measures that take into account the structure of the hierarchy as well; in [12] some measures are introduced to evaluate the lattice discovered by Subdue. In the present work we especially refer to measures related to the cardinality of a cluster, representing the number of times the related substructure occurs in the input graph, namely *frequency* and *representativeness* (REP). The main difference between the two measures is that the latter does not consider repetitions of substructures in a graph, measuring the number of input graphs holding the given substructure at least once. For example, given a simple dataset of two graphs G={($A\rightarrow A$); ($A\rightarrow A \rightarrow C \rightarrow A \rightarrow A$)} and the lattice in Figure 1, the REP of the cluster C_1 is 2 and its

Table 1: Dataset characteristics

# instances	# act	min # act/proc	max # act/proc	avg # act/week
24	693	3	253	29

frequency is 3. Both these measures have to be taken in proper account when the hierarchical cluster is used for information retrieval tasks.

4 Experiments

In this Section we show some experimental results obtained from the log of our case study. We'd like to point out that since activities in the log are described with very low-level details, we assigned each activity to a class. Hence, a class is an aggregation of low-level events describing them with a higher level of abstraction. In particular, we identified the following classes: a) *articleCreation, articleDeleting, articleUpdate* to represent activities regarding the paper writing, b) *codeCreation, codeDeleting, codeUpdate* for activities related to code editing, c) *Chat, emailSending* for messaging activities, i.e. Skype chat and emails respectively. Interested readers can find a more detailed description of the log preprocessing in [6]. In order to have different process instances, we split the log of our case study, that is related to only one paper, in 24 process instances on the basis of weeks; each of them was delimited by two artificial events representing the "Start" and the "End" of the week. Table 1 shows the characteristics of the dataset. There are 693 activities on the whole dataset spread over the 24 weeks. The shortest process is made of just 3 activities including the "Start" and the "End", while the longest has 253 activities, and there are 29 activities per week on average.

In the following Subsections we show the results obtained both by means of a classical process mining technique and by the Subdue algorithm. Then, we discuss and compare such outcomes, pointing out main benefits and drawbacks of both approaches.

4.1 Fuzzy Miner

Fuzzy Miner[10] is a process mining algorithm commonly used for processes with little or no structure (as those we consider), because it is aimed at extracting the main process behaviour rather than the precise process schema, usually too detailed to be useful. The algorithm outcome is a schema involving just the most relevant events (i.e. schema nodes), displayed as single events or aggregated in clusters, and their sequences (i.e. schema edges). The algorithm exploits two parameters: *significance* and *correlation*. The former represents both the importance of each event and of the events sequences, the latter how closely related are subsequent events. The least relevant events are simply deleted from the final output or, if they belong to a set of highly correlated events, they are clustered

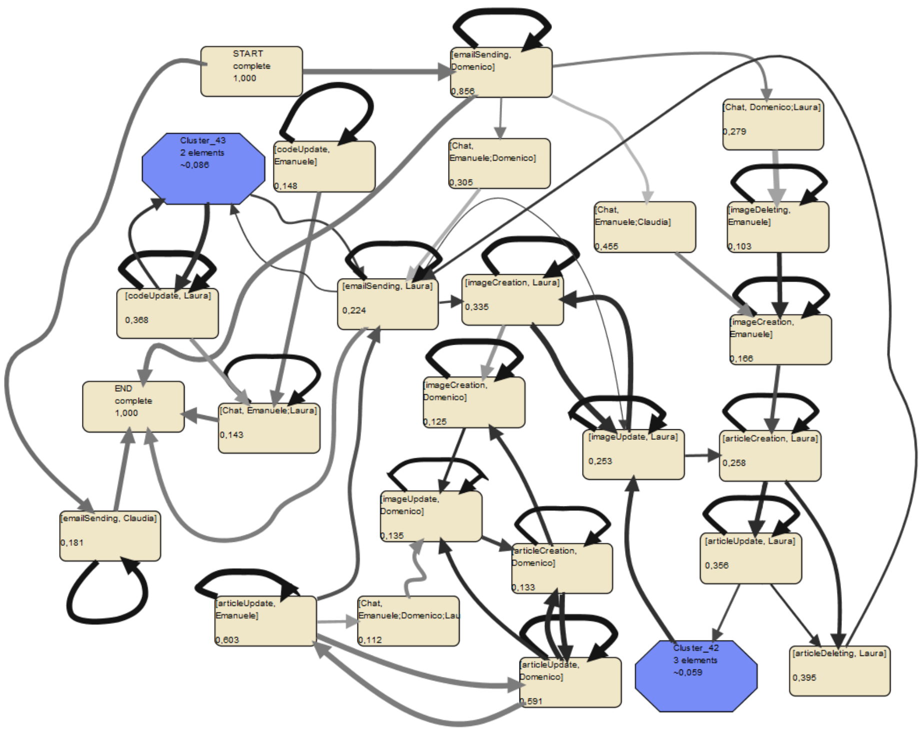

START complete 1,000
[emailSending, Domenico] 0,856
Cluster_43 2 elements ~0,086
[codeUpdate, Emanuele] 0,148
[Chat, Emanuele;Domenico] 0,305
[Chat, Domenico;Laura] 0,279
[Chat, Emanuele;Claudia] 0,455
[imageDeleting, Emanuele] 0,103
[codeUpdate, Laura] 0,368
[emailSending, Laura] 0,224
[imageCreation, Laura] 0,335
[imageCreation, Emanuele] 0,166
END complete 1,000
[Chat, Emanuele;Laura] 0,143
[imageCreation, Domenico] 0,125
[imageUpdate, Laura] 0,253
[articleCreation, Laura] 0,258
[emailSending, Claudia] 0,181
[imageUpdate, Domenico] 0,135
[articleCreation, Domenico] 0,133
[articleUpdate, Laura] 0,356
[articleUpdate, Emanuele] 0,603
[Chat, Emanuele;Domenico;Lau 0,112
[articleUpdate, Domenico] 0,591
Cluster_42 3 elements ~0,059
[articleDeleting, Laura] 0,395

Fig. 2: The fuzzy miner outcome

in a single event. Similarly, least relevant sequences are not displayed in the outcome. The user can decide between several possible ways to compute significance and correlation, e.g. the *frequency* for the significance and the *proximity* for correlation, that is how temporally close two events are. Such a value is used to define a filter: only events and sequences whose values are greater than a given threshold will be used to define the final outcome. In our analysis we use the standard algorithm configuration implemented in ProM Framework[1].

Figure 2 shows the outcome obtained by the Fuzzy Miner in our case. The square nodes represent single events types (i.e. different event id), while edges represent sequences of events; the significance of an edge is represented by its gray level, i.e. the more significant the edge is, the darker the corresponding arc in the graph is. The significance of a node is reported under the event name. Octagonal nodes represent clusters of low-significant and highly correlated events, containing the number of events belonging to the cluster and their mean significance.

[1] http://www.promtools.org/prom6/

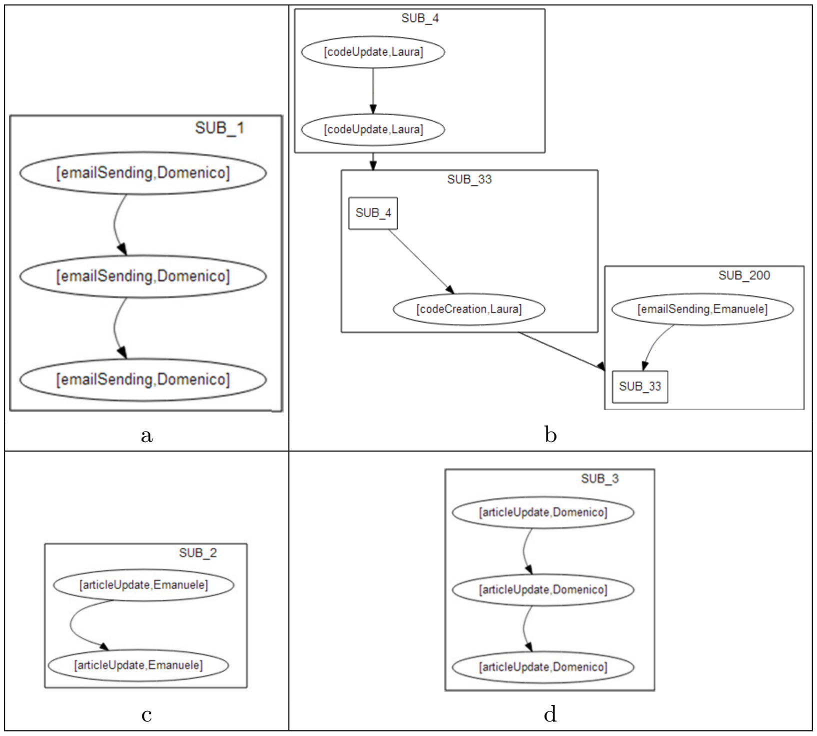


Fig. 3: The top four discovered substructures

4.2 Subdue

By using the Subdue algorithm, we obtained as a result a lattice formed by 245 substructures, with 49 top-level substructures (i.e. the 20%); the first ten top-level substructures have a representativeness of 22.1% on average. Note that in order to not affect the result of the experiment in favour of Subdue, we have just used the default parameters settings, as in the case of Fuzzy Miner.

Figure 3 shows the top four substructures obtained by the algorithm.

4.3 Discussion

Both previous approaches allow us to derive some interesting knowledge about the process. However, they have different aims, consisting respectively in deriving a complete process schema, for Fuzzy Miner, and in extracting the most significant patterns for Subdue. Therefore, the first technique is the most suitable if one is interested in exploring the activity flow as a whole, since it provides an overview of the entire process as shown in Figure 2. Nevertheless, the analysis of the outcome allows us to uderstand also significant behaviours in the process. For instance, we can check if some of the members assumed a key role in conducting one of the activities; in our case, in the left section of the graph, we can note that code editing operations involved only Laura and Emanuele. Moreover, there are not significant relationships between code operations performed by Emanuele and those performed by Laura; such nodes are mostly connected to messaging nodes. Such configuration suggests that the two members mostly worked on distinct code parts, coordinating each other by emails or Skype. Similarly, we can derive that the member most involved in email exchange was Domenico.

It is noteworthy that in Fuzzy Miner's outcome the discovery of significant patterns is not straightforward: it requires further processing, i.e. removal of activities and edges having significance lower than a manually chosen threshold. On the contrary, Subdue is aimed at identifying the most frequent and relevant substructures, hence highlighting the most significant patterns. Clearly, in such a way we lose the overall vision about the process. As a consequence, in order to gain knowledge regarding particular process aspects, like the ones discussed in the previous example, several substructures have to be explored. For instance, in Figure 3, code editing operations are in SUB_4, whose actor is only Laura. Then, in order to find the communication with Emanuele we need to explore the SUB_4 sub-lattice (3.b) towards the SUB_{200}. Since the representativeness of a substructure is always less than or equal to the representativeness of its parents, we can derive that such a collaborative pattern is actually not very relevant in this context. Moreover, by looking for code editing operations performed by Emanuele, we were able to find them only in the SUB_{26}, thus suggesting us that code editing operations were more frequently performed by Laura.

An interesting result obtained by using Subdue concerns the team work organizations. Indeed, by exploring the first SUBs, we mostly found simple substructures regarding actions performed by a single member, like SUB_2, SUB_3 represented in 3.c and 3.d. The coordination activities are described only in lower level substructures. This reveals that a relevant trend for this case study consists in a well-defined work division, where each member was involved in specific parts of paper development. This can suggest the need of some actions aimed to enrich cooperation between team members.

Finally, we'd like to focus on SUB_1. Such a substructure presents a quite surprising behaviour, namely that one of the members usually sent at least three emails one after the other during the analysed period. Since it seemed an anomalous behaviour, we explored our log to identify possible causes for the presence of this substructure. By doing so, we discovered that the pattern is present in weeks between the first submission and the acceptance notification, when paper and code editing activities were stopped, and then activities related to the organization of a satellite workshop organized by team members emerged. In particular they refer to workshop advertising and authors notification patterns that clearly justify multiple separate emails sending. Although related to the conference, this pattern can be regarded as noise with respect to the focus of paper development, suggesting further preprocessing in order to remove emails related to the workshop. In other words, the approach can also detect anomalous behaviours, thus originating an iterative process of event log improving. Note that we cannot derive this anomaly from the Fuzzy Miner outcome.

Before closing this Section, we'd like to draw some considerations about the data used in our case study. As already mentioned, we took into account only one scientific paper, splitting its activities on the basis of weeks; in such a way we obtained process instances quite different from each other. In particular, we observed that activities about paper editing and code editing were mainly performed in different weeks; therefore, it is very unlikely that we can extract complex shared patterns by considering weekly activities distributions. We can note that the average representativeness of the first top ten substructures allows us to estimate the overall quality of the model, also without exploring the derived patterns. In fact, we have obtained a value of 20.1%, which means that we have to expect not very relevant substructures.

Results show that the applied technique is actually able to aid users in process analysis, returning at least the same information extracted by a schema discovery technique and resulting more suitable to explore collaborative patterns. We can likely figure out that, if the technique were applied over more process instances, more significant patterns could be inferred, that can reveal us common working practices.

5 Conclusions and future work

In the present work, we discussed a methodology aimed at deriving relevant collaboration patterns belonging to unstructured processes, usually performed in knowledge-intensive domains. To this end, we exploit a hierarchical clustering technique, that is able to extract relevant patterns representing valuable knowledge about the collaboration process. To validate our approach, we applied it to a real case study, regarding the development of a scientific paper; we also compared the obtained results with the outcome of a well-known process mining technique. In such a way, we highlighted some benefits of our methodology with respect to the schema discovery approach, e.g. the focus on relevant substructures, not immediately available or not considered by exploiting such kind of approach.

However, we also pointed out that in our case study we couldn't derive very complex patterns, mainly because we considered only one paper, thus limiting the algorithm in extracting complex common behaviours. Currently, we are extending our data collection, in order to consider more different cases. In such a way, we expect to be able to derive patterns that represent the general team working habits as regards paper development. Another interesting issue regards the detection of parallel branches in events sequence. By now, we just take into account the temporal sequences of events; in future works we intend to consider the presence of possible parallelism in team members' actions, enabling us to discover more significant collaboration patterns.

References

1. van der Aalst, W.M.P.: Discovery, Conformance and Enhancement of Business Processes. Springer Berlin Heidelberg, Berlin (2011)

2. Barros, B., Verdejo, M.F.: Analysing student interaction processes in order to improve collaboration. The DEGREE approach. International Journal of Artificial Intelligence in Education 11(3), 221–241 (2000)
3. Jagadeesh Chandra Bose, R.P., van der Aalst, W.M.P.: Trace clustering based on conserved patterns: Towards achieving better process models. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) Business Process Management Workshops, LNBIP, vol. 43, pp. 170–181. Springer Berlin Heidelberg (2010)
4. Bratitsis, T., Dimitrakopoulou, A.: Data recording and usage interaction analysis in asynchronous discussions: The DIAS system. In: 12th International Conference on Artificial Intelligence in Education AIED, Workshop "Usage Analysis in Learning Systems". pp. 17–24 (2005)
5. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive processes: An overview of contemporary approaches. In: International workshop on Knowledge-intense Business Processes. pp. 32–47 (2012)
6. Diamantini, C., Genga, L., Potena, D.: A methodology for building log of collaboration processes. In: The 2014 International Conference on Collaborative Technologies and Systems. pp. 337–344. IEEE Press (2014)
7. Diamantini, C., Potena, D., Storti, E.: Mining usage patterns from a repository of scientific workflows. In: 27th Annual ACM Symposium on Applied Computing. pp. 152–157. ACM Press (2012)
8. Greco, G., Guzzo, A., Ponieri, L., Sacca, D.: Discovering expressive process models by clustering log traces. IEEE Transactions on Knowledge and Data Engineering 18(8), 1010–1027 (2006)
9. Gronau, N., Weber, E.: Management of knowledge intensive business processes. In: Desel, J., Pernici, B., Weske, M. (eds.) Business Process Management, LNCS, vol. 3080, pp. 163–178. Springer Berlin Heidelberg (2004)
10. Günther, C.W., van Der Aalst, W.M.P.: Fuzzy mining-adaptive process simplification based on multi-perspective metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) Business Process Management, LNCS, vol. 4714, pp. 328–343. Springer Berlin Heidelberg (2007)
11. Huo, M., Zhang, H., Jeffery, R.: A systematic approach to process enactment analysis as input to software process improvement or tailoring. In: Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific. pp. 401–410. IEEE Press (2006)
12. Jonyer, I., Cook, D.J., Holder, L.B.: Graph-based hierarchical conceptual clustering. The Journal of Machine Learning Research 2, 19–43 (2002)
13. Jung, J.Y., Bae, J., Liu, L.: Hierarchical business process clustering. In: 2008 IEEE International Conference on Services Computing. vol. 2, pp. 613–616. IEEE Press (2008)
14. Lipponen, L.: Exploring foundations for computer-supported collaborative learning. In: Stahl, G. (ed.) Computer support for collaborative learning: Foundations for a CSCL community. Proceedings of the Computer-Supported Collaborative Learning 2002 Conference. pp. 72–81. International Society of the Learning Sciences (2002)
15. Rubin, V., Günther, C.W., van der Aalst, W.M.P., Kindler, E., van Dongen, B.F., Schäfer, W.: Process mining framework for software processes. In: Wang, Q., Pfahl, D., Raffo, D. (eds.) Software Process Dynamics and Agility, LNCS, vol. 4470, pp. 169–181. Springer Berlin Heidelberg (2007)
16. Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace clustering in process mining. In: Ardagna, D., Mecella, M., Yang, J. (eds.) Business Process Management Workshops, LNBIP, vol. 17, pp. 109–120. Springer Berlin Heidelberg (2009)

A Framework for Learning Knowledge-Powered Word Embedding

Qing Cui[1], Bin Gao[2], Jiang Bian[2], Siyu Qiu[3], and Tie-Yan Liu[2]

[1] Dept. of Mathematical Sciences, Tsinghua University, Beijing, 100084, P. R. China,
cuiq12@mails.tsinghua.edu.cn
[2] Microsoft Research, 13F, Bldg 2, No. 5, Danling St, Beijing, 100080, P. R. China,
{bingao, jibian, tyliu}@microsoft.com
[3] Nankai University, Tianjin, 300071, P. R. China,
ppqq2356@gmail.com

Abstract. Neural network techniques are widely applied to obtain high-quality distributed representations of words, i.e., word embeddings, to address text mining and natural language processing tasks. Recently, efficient methods have been proposed to learn word embeddings from context that captures both semantic and syntactic relationships between words. However, it is challenging to handle unseen words or rare words with insufficient context. In this paper, we propose a framework that can leverage general pairwise word similarity to address these challenges. As an example, we propose to take advantage of seemingly less obvious but essentially important morphological word similarity to show the power of our framework. In particular, we introduce a novel neural network architecture that leverages both contextual information and morphological word similarity to learn word embeddings. Experiments on an analogical reasoning task demonstrates that the proposed method can greatly enhance the effectiveness of word embeddings.

1 Introduction

Deep learning techniques have been widely applied to solve text mining and natural language processing (NLP) tasks, the basis of which yields obtaining high-quality distributed representations of words, i.e., word embeddings. In recent years, efficient methods, such as the continuous bag-of-word (CBOW) model and the continuous Skip-gram (Skip-gram) model, have been proposed to leverage the surrounding context of a word in documents to transform words into vectors (i.e., word embeddings) in a continuous space, which surprisingly captures both semantic and syntactic relationships between words. The underlying principle in these works lies in that words that are syntactically or semantically similar should have similar surrounding contexts.

While these works have demonstrated their effectiveness in various tasks, they also suffer from a couple of limitations. **(i)** It is difficult to obtain word embeddings for new words since they are not included in the previous vocabulary. Some previous studies [15] used a default index to represent all unknown words, but such a solution will inevitably lose information for emerging words. **(ii)** The embeddings for rare words are unreliable due to the insufficient surrounding contexts. Since the aforementioned works adopt statistical methods, when a word has only a few occurrences in the training data, they will fail in extracting statistical clues to correctly map the word into the embedding space.

In sharp contrast, according to the studies on word recognition in cognitive psychology [9,8], when a human looks at a word, no matter new or rare, he/she can figure out effective ways to understand it. For instance, one sometimes conducts phonological recoding through blending graphemes into phonemes and blend syllabic units into recognizable words; one may also analyze the root/affix of the new word so as to build its connections with his/her known words. Suppose the new word is *inconveniently*. Given its root/affix, i.e., *in-convenient-ly*, it is natural to guess that it is the adverb form of *inconvenient* and the latter is probably the antonym of *convenient*. Henceforth, morphological word similarity can act as an effective bridge for understanding new or rare words based on known words in the vocabulary. Inspired by this word recognition process, we propose using morphological information to enhance the deep learning framework for word embedding. In particular, beyond the contextual information already used in CBOW and Skip-gram, we take advantage of morphological similarity between words in the learning process so as to handle new or rare words.

Although the morphological knowledge contains invaluable information, it might be risky to blindly rely on it. The reason is that the prediction based on morphological similarity is somehow only a kind of guess, and there exist many counter examples inconsistent with it. For example, if only looking at the morphological similarity, one may link *convention* to *convenient* since they share a long substring. However, it is clear that these two words are neither syntactically nor semantically similar. In this case, if we stick to the morphological knowledge, the effectiveness of the learned word embeddings could be even worse. To tackle this issue, we once again leverage the findings regarding word recognition in cognitive psychology [9,8]. It has been revealed that humans can take advantage of the contextual information (both the context at the reading time and the context in his/her memory) to correct the unreliable morphological word similarity. By comparing their respective contexts, one can distinguish between *convenient* and *convention* and weaken the morphological connection between these two words in his/her mind. Inspired by this, we also propose updating the morphological knowledge during our learning process. Specifically, we will not fully trust the morphological knowledge, and will change it so as to maximize the consistency between contextual information and morphological word similarity.

To sum up the discussions above, we actually develop a novel neural network architecture that can leverage morphological word similarity for word embedding. Our proposed model consists of a contextual information branch and a morphological knowledge branch. On one hand, we adopt the state-of-the-art Skip-gram model [17] as our contextual information branch for its efficiency and effectiveness. On the other hand, we explore edit distance, longest common substring similarity, morpheme similarity, and syllable similarity as morphological knowledge to build a relation matrix between words, and put the relation matrix into the morphological knowledge branch. These two branches share the same word embedding space, and they are combined together using tradeoff coefficients in order to feed forward to the output layer to predict the target word. The back propagation stage will modify the tradeoff coefficients, word embeddings, and the weights in the relation matrix layer by layer. We have conducted experiments on a publicly available dataset, and the results demonstrate that our proposed approach can help produce improved word representations as compared with the state-of-the-art methods on an analogical reasoning task.

2 Related Work

Word embedding as continuous vectors has been studied for a long time [12]. Recently, deep learning methods have been applied to obtain continuous word embeddings to solve a variety of text mining and natural language processing tasks [4,10,16,17,23,25,26,7,5,18,24]. [4]For example, Collobert *et al* [4,5] proposed a unified neural network architecture that learns word representations based on large amounts of unlabeled training data, to deal with several different natural language processing tasks. Mikolov *et al* [16,17] proposed the continuous bag-of-words model (CBOW) and the continuous skip-gram model (Skip-gram) for learning distributed representations of words also from large amount of unlabeled text data; these models can map the semantically or syntactically similar words to close positions in the word embedding space, based on the intuition that the contexts of the similar words are similar. All the above work does not leverage rich extra knowledge when learning word embeddings.

There are some knowledge related word embedding works in the literature, but most of them were targeted at the problems of knowledge base completion and enhancement [3,22,27] rather than producing high-quality word embeddings, which is different with our work. Besides, Luong *et al* [14] proposed a morphological Recursive Neural Network (morphoRNN) that combines recursive neural networks and neural language models to learn better word representations, in which they regarded each morpheme as a basic unit and leveraged neural language models to consider contextual information in learning morphologically-aware word representations.

However, it only focused on the morphological structure inside a word, but did not consider the morphological similarity between words. Hence, some morphological knowledge like the edit distance and the longest common substring cannot be used in its framework. In this paper, we propose a novel neural network architecture that can leverage any kind of pairwise word similarity.

3 Word Embedding Powered by Morphological Knowledge

3.1 New Neural Network Architecture

In this subsection, we describe our proposed new neural network architecture that leverages both contextual information and morphological knowledge to learn word embedding. We use Skip-gram [17] as a baseline model to illustrate how our framework works.

[4] The deep learning methods significantly outperforms traditional methods such as Latent Semantic Analysis (LSA) [16,19], so we focus on comparison with deep learning methods in this paper.

Skip-gram The Skip-gram model aims to learn the latent word representations that are good at predicting the surrounding words in the training text stream. Given a sequence of training words $w_1, w_2, \ldots, w_T$, the objective of the Skip-gram model is to maximize the following average log probability,

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-N\leq j\leq N, j\neq 0}\log p(w_{t+j}|w_t),$$

where N is the size of context window. By using w_O to denote the output word, i.e., w_{t+j}, and using w_I to denote the input word, i.e., w_t, the conditional probability $p(w_{t+j}|w_t)$ is defined using the *softmax* function,

$$p(w_O|w_I) = \frac{\exp(v_{w_O}^{'}{}^{T} v_{w_I})}{\sum_w \exp(v_w^{'}{}^{T} v_{w_I})},$$

where v_w and $v_w^{'}$ are the *input* and *output* representation vectors of w, and the sum in the denominator is over all words in the vocabulary. It is difficult and impractical to directly optimize this objective because computing the derivative is proportional to the vocabulary size, which is often very large.

Several approaches [21,1,2] have been employed to tackle this problem. The state-of-the-art method is noise-contrastive estimation (NCE) [11], which aims at fitting unnormalized probabilistic models. NCE can approximate the log probability of *softmax* by performing logistic regression to discriminate between the observed data and some artificially generated noise. It was first adapted in the neural language model in [20], and was then applied to the inverse vector log-bilinear model [19]. Another (simpler) method is negative sampling (NEG), which generates k noise samples for each input word to estimate the objective.

By using NEG, the *softmax* conditional probability $p(w_{t+j}|w_t)$ will be replaced by

$$J(\theta) = \log\sigma(v_{w_O}^{'}{}^{T} v_{w_I}) + \sum_{i=1}^{k}\mathbb{E}_{w_i\sim P_n(w)}\left[\log\sigma(-v_{w_i}^{'}{}^{T} v_{w_I})\right]$$

where θ is the model parameter including the word embeddings; σ denotes the logistic function; and $P_n(w)$ represents the noise distribution which is set as the 3/4 power of the unigram distribution $U(w)$, i.e., $P_n(w) = U(w)^{3/4}/Z$. Then, we can estimate the gradient of $J(\theta)$ by computing

$$\frac{\partial J(\theta)}{\partial\theta} = (1-\sigma(v_{w_O}^{'}{}^{T} v_{w_I}))\frac{\partial v_{w_O}^{'}{}^{T} v_{w_I}}{\partial\theta} - \sum_{i=1}^{k}\left[\sigma(v_{w_i}^{'}{}^{T} v_{w_I})\frac{\partial v_{w_i}^{'}{}^{T} v_{w_I}}{\partial\theta}\right]$$

By summing over k noise samples instead of a sum over the entire vocabulary, the training time yields linear scale to the number of noise samples and becomes independent of the vocabulary size.

Our Model To incorporate morphological knowledge into the learning process, we propose a new neural network architecture. Beyond the basic Skip-gram model that predicts a target word based on its context, the proposed new method introduces a parallel branch that leverages morphological knowledge to assist predicting target word, as shown in Figure 1. Intuitively, when a word w_t is the center word in the context window, we predict the surrounding words by leveraging not only the representation of word w_t as contextual information (referred as *contextual information branch*) but also the representations of the words that are morphologically similar to w_t (referred as *morphological knowledge branch*).

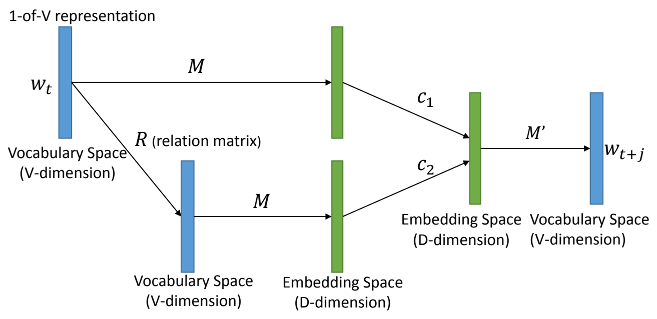

1-of-V representation
w_t
M
c_1
R (relation matrix)
M'
w_{t+j}
Vocabulary Space (V-dimension)
c_2
M
Embedding Space (D-dimension)
Vocabulary Space (V-dimension)
Vocabulary Space (V-dimension)
Embedding Space (D-dimension)

Fig. 1. The proposed neural network architecture

According to Figure 1, to obtain the representation of a center word w_t from the morphological knowledge branch, it is necessary to find the set of words morphologically similar to w_t, which is denoted as R_t. Then, we can extract the embedding of each word in R_t from the embedding matrix M shared with the contextual information branch. After that, a corresponding knowledge representation of R_t can be computed by feeding forward the relationship layer, which is written as

$$v_{R_t} = \sum_{w \in R_t} s(w_t, w) v_w,$$

where $s(w_t, w)$ is the similarity score, the methods of computing which will be introduced in Section 3.2. Actually v_w is the i-th row of matrix M where i is the index of the word w in the vocabulary, and $s(w_1, w_2)$ is the element of relation matrix R at (i, j) which are the indices of words w_1, w_2 in the vocabulary respectively. To ensure the quality of morphological knowledge and control the number of parameters, we only leverage the top words with highest morphological similarity scores as R_t. In our model, an input word can only connect to at most five words in the relationship layer. This sparse structure will not change during training, and only the weights of these connections will be updated. Therefore, we will not suffer from a huge number of parameters even if R is learned.

Finally, an aggregated representation of the input word, denoted as v_{w_I}, can be calculated as the weighted sum of the representations from the contextual information branch and the morphological knowledge branch, i.e.,

$$v_{w_I} = c_1(w_t) v_{w_t} + c_2(w_t) v_{R_t},$$

where $c_1(\cdot)$ and $c_2(\cdot)$ are the functions of w_t and yield much dependency on the word frequency. Intuitively, frequent words are associated with much more training samples than rare words, such that it is easy to collect rich contextual information for frequent words, while the contextual information for rare words might be insufficient. In contrast, the volume of morphological knowledge of a word usually has little correlation to the word frequency, thus, rare words can still rely more on the morphological knowledge even though the contextual information is not reliable. Therefore, the balancing function $c_1(\cdot)$ and $c_2(\cdot)$ should be related to word frequency. Specifically, we divide the words into a number of buckets according to their frequencies, and all the words in the same bucket will share the same values of $c_1(\cdot)$ and $c_2(\cdot)$.

A more explicitly intuitive way to interpret the above model is as follows. For each word w_t, we use one row in the embedding matrix M to encode its contextual embedding. In addition, by using matrix R, we can identify a couple of morphologically similar words to w_t. Then we can also extract the contextual embeddings of these similar words from M and take the weighted average of these embedding vectors as the morphological embedding for the original word w_t. Then finally the overall embedding of w_t is computed as the weighted combination of its contextual embedding and morphological embedding. Matrix M' is used to predict the surrounding word w_{t+j} based on the overall embedding of w_t. In our model, the parameters to train include M, R, $M^{'}$, and multiple pairs of c_1 and c_2 (corresponding to different frequency buckets). In our implementation, we learn these parameters with negative sampling, standard back propagation and gradient descent.

3.2 Morphological Knowledge

As compared to Skip-gram, the uniqueness of our model lies in the introduction of the morphological knowledge branch. In this subsection, we will make discussions on how to realize this new branch. In particular, we propose four types of naturally defined morphological knowledge. Note that this is not a complete study on morphological knowledge, but we can use these four specific types as examples to show the effectiveness of the proposed framework. Any other types of morphological knowledge can be used under our proposed framework.

Edit Distance Similarity (Edit). Edit distance is a way of quantifying how dissimilar two strings (e.g., words) are by counting the minimum number of operations required to transform one string into the other. The operations might be *letter insertion*, *letter deletion*, or *letter substitution*. We calculate the edit distance similarity score for two words w_1 and w_2 as $s_{Edit}(w_1, w_2) = 1 - \frac{d(w_1,w_2)}{\max(l(w_1),l(w_2))}$, where $d(w_1, w_2)$ represents the edit distance of the two words and $l(w_1), l(w_2)$ are the corresponding word lengths.

Longest Common Substring Similarity (LCS). Longest common substring similarity is defined as the ratio of the length of the longest shared substring of two words (denoted by $g(w_1, w_2)$) and the length of the longer word, i.e., $s_{LCS}(w_1, w_2) = \frac{g(w_1,w_2)}{\max(l(w_1),l(w_2))}$.

Morpheme Similarity (Morpheme). Morpheme similarity is calculated based on the shared root (or stem) and affix (prefix and suffix) of two words. Suppose each word of w_1 and w_2 can be split into a set of morphemes (denoted by $F(w_1)$ and $F(w_2)$), then the morpheme similarity of the two words is calculated as $s_{Morpheme} = \frac{|F(w_1)\bigcap F(w_2)|}{\max(|F(w_1)|,|F(w_2)|)}$, where $|\cdot|$ outputs the size of the set.

Syllable Similarity (Syllable). Syllable similarity is calculated based on the shared syllables of two words. Suppose both w_1 and w_2 can be split into a set of syllables (denoted by $G(w_1)$ and $G(w_2)$), then the syllable similarity of the two words is calculated as $s_{Syllable} = \frac{|G(w_1) \bigcap G(w_2)|}{\max(|G(w_1)|, |G(w_2)|)}$.

In addition to using these four types of morphological word similarity separately, one can also combine them together. In the next section, we will conduct experimental study on all these different choices.

4 Experiments

4.1 Experimental Setup

Datasets The training set used in our experiments is the *enwik9* data[5], which is built from the first billion characters from Wikipedia. This corpus contains totally 123.4 million words. We used Matt Mahoney's text pre-processing script[6] to process the corpus. After pre-processing, all digits were replaced with English words (e.g., *3* was replaced with *three*), and the metadata and hyperlinks were removed. Furthermore, all words that occurred less than 5 times in the training data were discarded from the vocabulary, resulting in a vocabulary of 220 thousand words. The out-of-vocabulary words were ignored in training.

Compared Methods and Experimental Settings In our experiments, we compare our proposed methods with two baselines:

• **Skip-gram** (baseline): is a popular model as introduced by [17].

• **Skip-gram + Edit/LCS/Morpheme/Combination Input Feature** (baseline): Another baseline uses the morphological features as additional inputs during training of the Skip-gram model. Specifically, the input is no longer a 1-of-V representation but will append the morphological feature which is the corresponding row of the relation matrix R. Thus the input is a vector of length $2V$ and the projection matrix have the size of $2V \times D$ where D is the dimension of word embeddings. In our experiments, we employed four types of morphological knowledge. Edit and LCS can be computed directly from the definitions. For Morpheme, we used a public tool called Morfessor [6], which can split a word into morphological segments with prefix, stem, and suffix tags. For Syllable, we implemented the hyphenation tool proposed by Liang [13], which has been used in many editing softwares like LaTeXto break words by syllables. Moreover, we also test the performance by combining these three types of knowledge features into a union feature set.

• **Skip-gram + Edit/LCS/Morpheme/Combination Relation Matrix**: In our model, we employ the same types of morphological knowledge. For each of them, given a word w, we calculated its similarities to all the other words and selected the top 5 words with highest similarity to build the relation edges in the weight matrix R.[7] We tested the R matrix built based on each single type of knowledge, and we also tested the R matrix built based on several types of knowledge through combination. Specifically, given the

[5] http://mattmahoney.net/dc/enwik9.zip

[6] http://mattmahoney.net/dc/textdata.html

[7] In our experiments, the performance varies little when the number of similar words varies from 3 to 50.

four ranked lists of words from the morphological knowledge, we combined them into a union set, and selected the top 5 words that got more votes by the four knowledge types. Note that, our model can be degraded to the second baseline by fixing $c1$, $c2$, M and not sharing M.

In all these three methods, we set the dimension of word embeddings to 100 and the context windows size to 5; we employed the negative sampling technique to train both models and the number of negative samples was set to 3.

As discussed in Section 3.1, the balancing parameters in our proposed model might be related to word frequency. For simplicity, we used a greedy algorithm to divide the words into a certain number of buckets. Specifically, suppose we want to have b buckets, then we rank the words in the vocabulary by their frequencies in the descending order, and put the words into the first bucket one by one until the summed frequency of the first bucket reaches the $1/b$ of the total frequency; then we feed the rest buckets in the similar way, and eventually the summed frequency of each of the b buckets is approximately equal to $1/b$ of the total frequency. We let all words in one bucket share the same balance coefficients. In our experiments, we set the bucket number to 1000.

With the above settings, the training time of the proposed model was only about 1.5 times of the original Skip-gram model, showing that our neural network framework is very efficient. It can finish less than 15 minutes on a single machine with four cores.

4.2 Evaluation Tasks

Analogical Reasoning Task The analogical reasoning task was introduced by Mikolov *et al* [16]. The task consists of 19,544 questions of the form "a is to b is as c is to _", denoted as $a : b \rightarrow c : ?$. Suppose $\overrightarrow{w}$ is the learned word representation vector of word w normalized to unit norm. Following [16], we answer this question by finding the word d^* whose representation vector is the closest to vector $\overrightarrow{b} - \overrightarrow{a} + \overrightarrow{c}$ according to cosine similarity excluding b and c, i.e., $d^* = \arg\max_{x \in V, x \neq b, x \neq c}(\overrightarrow{b} - \overrightarrow{a} + \overrightarrow{c})^T \overrightarrow{x}$. The question is regarded as answered correctly only when d^* is exactly the answer word in the evaluation set. There are two categories in the task, with 8,869 semantic analogies (e.g., *England* : *London* $\rightarrow$ *China* : *Beijing*) and 10,675 syntactic analogies (e.g., *amazing* : *amazingly* $\rightarrow$ *unfortunate* : *unfortunately*).

4.3 Experimental Results

Comparison between the Two Baselines Table 1 compares the performance of two baselines. From the table, we can observe that the Skip-gram model is the best of the six models which implies that simply adding morphological knowledge as additional input features doesn't work. The reason may be that the morphological knowledge is very noisy and we can barely benefit from it by blindly rely on it. This conclusion can be confirmed by the more careful comparison between different knowledge. While Edit and LCS are stricter than Morpheme and Syllable especially when we only pick the top 5 most similar words, the performance of Edit and LCS are always better than Morpheme and Syllable. In the following experiments, we only compare our proposed model with the Skip-gram model.

Table 1. Comparison between the baselines on analogical reasoning task. We report the semantic/syntactic/total accuracy in analogical reasoning task.

Model	Analogical Reasoning Task		
	Semantic	Syntactic	Total
Skip-gram	21.85%	36.64%	28.84%
+ Edit Input Feature	13.67%	27.85%	21.41%
+ LCS Input Feature	13.65%	28.30%	21.65%
+ Morpheme Input Feature	13.55%	23.66%	19.07%
+ Syllable Input Feature	11.94%	25.30%	19.24%
+ Combination Input Feature	13.67%	28.52%	21.78%

Table 2. Performance of leveraging morphological knowledge on the analogical reasoning task.

Model	Semantic Accuracy	Gain	Syntactic Accuracy	Gain	Total Accuracy	Gain
Skip-gram	21.85%	-	34.64%	-	28.84%	-
+ Edit Relation Matrix	23.59%	+7.96%	43.49%	+25.55%	34.46%	+19.49%
+ LCS Relation Matrix	23.70%	+8.47%	44.50%	+28.46%	35.06%	+21.57%
+ Morpheme Relation Matrix	24.86%	+13.78%	43.68%	+26.10%	35.14%	+21.84%
+ Syllable Relation Matrix	24.53%	+12.27%	41.88%	+20.90%	34.01%	+17.93%
+ Combination Relation Matrix	23.58%	+7.92%	46.90%	+35.39%	36.32%	+25.94%

Results on Analogical Reasoning Task Table 2 demonstrates the performance of Skip-gram + Edit/LCS/Morphene/Syllable/Combination Relation Matrix on the analogical reasoning task. From this table, we can observe that:

(i) Adding morphological knowledge, either single type or combined knowledge, to the Skip-gram model can consistently increase all types of accuracies in the analogical reasoning task. This shows that morphological knowledge can improve the quality of the learned word embeddings.

(ii) Morpheme performs the best among the four types of knowledge in terms of total accuracy. We hypothesize the reason as that morphemes (like root and affix) are basic units in word composition, and it implies accurate syntactic and semantic correlation if two words share the same root. LCS achieves the second highest accuracy. The possible reason is that more than half of the test examples belong to syntactic word groups (like the pairs of adjective and adverb) where it is common to have long shared substrings between words. Edit performs a little worse, since letters themselves barely carry on syntactic and semantic information such that many words with different meanings can yield short edit distances. Syllables focus more on the pronunciation. While some syllables are overlapped with morphemes (like *re*, *im*), many others (like *ti*, *ta*) do not yield specific meanings and are likely to introduce much noise into the relation matrix so as to hurt the performance.

(iii) The combination of the four types of morphological knowledge can further improve the total accuracy, which indicates that every single type of morphological knowledge has its own limitations, and combining them together will significantly increase the recall of truly similar words.

(iv) The average gain on syntactic similarity (27.28%) is much higher than that on the semantic accuracy (10.08%), because morphological knowledge is more likely to

Table 3. Performance of leveraging morphological knowledge on the analogical reasoning task if we do not update the weights in the relation matrix in the learning process.

Model	Semantic Accuracy	Gain	Syntactic Accuracy	Gain	Total Accuracy	Gain
Skip-gram	21.85%	-	34.64%	-	28.84%	-
+ Edit Relation Matrix	21.42%	-1.97%	40.62%	+17.26%	31.91%	+10.64%
+ LCS Relation Matrix	23.48%	+7.46%	41.24%	+19.05%	33.18%	+15.05%
+ Morpheme Relation Matrix	23.94%	+9.57%	41.04%	+18.48%	33.28%	+15.40%
+ Syllable Relation Matrix	22.48%	+2.88%	40.61%	+17.23%	32.38%	+12.27%
+ Combination Relation Matrix	21.17%	-3.11%	43.60%	+25.87%	33.42%	+15.88%

capture syntactic information rather than semantic information between words, and many semantically similar words in this task (like *England* and *London*) might not appear similar in their morphological shapes.

Note that in our proposed model the morphological knowledge is just used to initialize the relationship matrix, and the non-zero values in the matrix will be updated during the learning process. This is coherent with the human cognitive psychology that blindly sticking to the morphological knowledge may even hurt in some cases. To verify this, we conducted some additional experiments where the relationship matrix is fixed during the learning process. The results are shown in Table 3. From this table, we can see that without updating the relationship matrix, the average improvements against the baseline method for any of methods are much smaller than those reported in Table 2. This clearly verifies our hypothesis and indicates the necessity to update the relation weights to maximize its consistency with the contextual information.

Table 4. Top five similar words in the embedding spaces powered by all 4 combination of morphological knowledge.

Example Word	Skip-gram	Combined Knowledge	Skip-gram + All 4 Combination
uninformative	monotherapy	informative	problematic
	lcg	inchoative	fallacious
	electrodeposition	inoperative	inaccurate
	astrophotography	interrogative	uninteresting
	ultrafilters	formative	precisely
stepdaughter	grandaughter	daughters	grandaughter
	swynford	daughter	daughter
	caesaris	grandaughter	daughters
	theling	steptoe	wife
	stepson	slaughter	stepfather
uncompetitive	overvalued	competitively	competitive
	monopsony	competitive	noncompetitive
	skyrocketing	noncompetitive	profitable
	dampened	competitiveness	competetive
	undervalued	competetive	lucrative

4.4 Case Study

To further understand our proposed framework, we sampled some rare words and checked the closest words to them in different word embedding spaces. Specifically, for a given word, we extracted its representation vector in the 100-dimension embedding space, and calculated its cosine similarity with the representation vectors of all the other words. Then we show the five most similar words generated by the methods under investigation in Table 4. According to Table 2, the combination of four types of knowledge achieved the best performance, therefore we only show the results for the baseline method (Skip-gram) and the combination method (Skip-gram + All 4 Combination). Beside, we also show the most similar words directly given by the four types of knowledge without going through the learning process (denoted as Combined Knowledge), which can give us an overview of how the original morphological knowledge looks like. Note that actually the baseline does a good job on frequent words and the results of our model on those words are similar to the baseline, so we only sampled some rare words to demonstrate the power of our model.

From Table 4, we have the following observations: **(i)** We can see that the Skip-gram method often fails in finding reasonable semantically or syntactically related words for rare words. For example, *uninformative* only appears 18 times in the training corpus, and thus its nearest neighbors are almost random. According to the morphological knowledge (see the column of Combined Knowledge), this word may have relation with *informative* and *formative*. By leveraging these relatively frequent words to enhance the embedding for *uninformative*, our model eventually generate very effective embedding for this rare word, and its similar words in the learned embedding space become much more reasonable. **(ii)** We can also see that the morphological knowledge could be noisy in some cases. For example, it suggests *inchoative* and *interrogative* to *uninformative*, because these words share a substring *ative* with *uninformative*. However, they are neither syntactically similar nor semantically similar. The power of our proposed framework lies in that it can distinguish useful knowledge and noise by seeking help from the contextual information, and refine the tradeoff coefficients and the relationship matrix to ensure the generation of a more reliable embedding. We can see that the most similar words to *uninformative* in the final embedding space, such as *problematic* and *inaccurate* are more semantically correlated to *uninformative* than *inchoative* and *interrogative*.

To sum up, the examples in Table 4 indicate that for rare words, (1) it is unreliable to learn their embeddings only from contexts; (2) morphological knowledge can do a great favor if we can successfully deal with the noise it brings in; (3) contextual information can help in distinguishing useful knowledge and noise. In this sense, our proposed framework achieves the goal of co-learning context and morphological knowledge to obtain high quality word embeddings.

5 Conclusions

We proposed a novel neural network framework to leverage morphological word similarity to learn high-quality word embeddings. The framework contains a contextual information branch to leverage word co-occurrence information and a morphological knowledge branch to leverage morphological relationship between words. We tested the framework on several tasks and the results show it can produce enhanced word representations compared with the state-of-the-art models.

For the future work, we plan to leverage other types of relationships (e.g., the relationships in the knowledge bases like *WordNet* and *Freebase*) in the proposed neural network framework to check whether we can obtain even better word representations.

References

1. Y. Bengio and J.-S. Senecal. Quick training of probabilistic neural nets by importance sampling, 2003.
2. Y. Bengio and J.-S. Senecal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *Trans. Neur. Netw.*, 19(4):713–722, 2008.
3. A. Bordes, J. Weston, R. Collobert, Y. Bengio, et al. Learning structured embeddings of knowledge bases. In *AAAI*, 2011.
4. R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, pages 160–167, New York, NY, USA, 2008. ACM.
5. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, 12:2493–2537, 2011.
6. M. Creutz and K. Lagus. Unsupervised models for morpheme segmentation and morphology learning. *TSLP*, 2007.

7. L. Deng, X. He, and J. Gao. Deep stacking networks for information retrieval. In *ICASSP*, pages 3153–3157, 2013.
8. L. C. Ehri. Learning to read words: Theory, findings, and issues. *Scientific Studies of reading*, 9(2):167–188, 2005.
9. L. C. Ehri, R. Barr, M. Kamil, P. Mosenthal, and P. Pearson. Development of the ability to read words. *Handbook of reading research*, 2:383–417, 1991.
10. X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML*, 2011.
11. M. U. Gutmann and A. Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *J. Mach. Learn. Res.*, 13:307–361, 2012.
12. G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Distributed representations. In *Parallel distributed processing: Explorations in the microstructure of cognition*, pages 3:1137–1155. MIT Press, 1986.
13. F. M. Liang. Word hy-phen-a-tion by com-put-er. Technical report, 1983.
14. M.-T. Luong, R. Socher, and C. D. Manning. Better word representations with recursive neural networks for morphology. *CoNLL-2013*, 104, 2013.
15. T. Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.
16. T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. ICLR '13, 2013.
17. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.
18. A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. In *NIPS*, pages 1081–1088, 2008.
19. A. Mnih and K. Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *NIPS*, pages 2265–2273. 2013.
20. A. Mnih and Y. W. Teh. A fast and simple algorithm for training neural probabilistic language models. In *ICML*, pages 1751–1758, New York, NY, USA, 2012. Omnipress.
21. F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *AISTATS*, pages 246–252, 2005.
22. R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, pages 926–934, 2013.
23. R. Socher, C. C. Lin, A. Y. Ng, and C. D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 2011.
24. J. P. Turian, L.-A. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *ACL*, pages 384–394, 2010.
25. P. D. Turney. Distributional semantics beyond words: Supervised learning of analogy and paraphrase. *TACL*, pages 353–366, 2013.
26. P. D. Turney and P. Pantel. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37:141–188, 2010.
27. J. Weston, A. Bordes, O. Yakhnenko, and N. Usunier. Connecting language and knowledge bases with embedding models for relation extraction. *arXiv preprint arXiv:1307.7973*, 2013.

Side Effects Analysis Based On Action Sets for Medical Treatments

Hakim Touati[1], Zbigniew W. Raś[1,2], James Studnicki[3], and Alicja A. Wieczorkowska[4]

[1] Univ. of North Carolina, College of Comp. and Informatics, Charlotte, NC 28223
[2] Warsaw Univ. of Technology, Inst. of Computer Science, 00-665 Warsaw, Poland
[3] Univ. of North Carolina, College of Health and Human Serv., Charlotte, NC 28223
[4] Polish-Japanese Institute of Information Technology, 02-008 Warsaw, Poland
`{htouati,ras,jstudnic,awieczor}@uncc.edu`

Abstract. Side effects result from the application of treatments to patients. They are commonly negative and therefore undesirable. Side effects are one of the major causes of readmissions in hospitals and generate additional expenses and poor healthcare quality. One of the main challenges in side effects study is their unexpectedness and the lack of predictability in a multi-factor environment. In this paper, we strive to extract negative side effects patterns from multivalued features. We also present a patients clustering scheme based on similar negative side effects (negative action sets). We evaluated our approach using the Florida State Inpatient Databases (SID), which is a part of the Healthcare Cost and Utilization Project (HCUP) [1]. Our results show that we are highly effective in extracting negative side effects.

Keywords: Side effects, Meta-actions, Action rules, Action sets, Action terms, Actionable knowledge

1 Introduction

Treatments' negative side effects discovery is a very challenging problem that has not been given a lot of attention from knowledge discovery researchers in the healthcare domain. Side effects are often resulting from the application of treatments modeled as Meta-actions [2]. Meta-actions represent actions triggering certain changes in objects' states that will execute action rules [3]. These changes are commonly referred to as meta-action effects that affect certain properties of the examined objects and are used for better personalization [4]. Meta-actions effects can be positive or neutral [5], and in some cases negative. Positive effects help objects positively to transition them into a more desired state. Neutral meaning does not introduce any effects on the overall state of the objects. Negative effects that may possibly harm or move the object into an undesired state. These effects can be seen as side effects when they are not intended by users.

There has been several work for treatment patterns recognition. Kolibaba et. al. [6] and Ramey et. al. [7] explored treatment patterns and outcomes. Lorigan et. al. [8] also explored treatment patterns and outcomes for patients with metastatic melanoma in the U.K. In [9], the authors mine medical articles for the disease and treatments as well as underlying side effects. However, they did not extract treatment negative side effects patterns in terms of results.

Treatment effects are not only related to the applied meta-actions, but also to the patient's initial state. In our previous work on extracting treatment effects, we extracted only positive and neutral side effects because we can analyze them based on the patient initial state. In fact, the union of the neutral and positive effects constitute the patient's initial state. However, treatment's side effects are unknown before applying the treatments. For this reason, it is important to study the objects in hand and anticipate the possible side effects when applying specific meta-actions. In this paper, we focus on the extraction and evaluation of potential negative side effects given an object and the applicable meta-actions. We also cluster patients based on the negative action sets extracted and evaluate them for the Florida State Inpatient Databases (SID) [1]. Our results show that action sets extracted are effectively the negative side effects for the treatments.

2 Background

In this section, we define the concepts used in actionable knowledge discovery that will help extract patterns of treatment effects. These concepts are used to extract any type of effects; however, we are interested in negative side effects.

Definition 1 (Information System) *By information system [10] we mean a triple of the form $S = (X, F, V)$ where:*

1. *X is a nonempty, finite set of objects.*
2. *F is a nonempty, finite set of features of the form $f : X \to 2^{V_f}$, which is a function for any $f \in F$, where V_f is called the domain of f.*
3. *V is a finite set of feature values such as: $V = \bigcup\{V_f : f \in F\}$.*

Definition 2 (Stable and Flexible features) *Stable features are object properties that we do not have control over in the context of an information system. For example, a birth date is a stable feature. On the other hand, flexible features are object properties that can transition from one value to another triggering a change in the object state. For instance, blood pressure is a flexible features.*

In the following, we will define the Atomic Action Terms that are the foundational expressions in actionable knowledge.

Definition 3 (Atomic action term in S) *also called elementary action term in S, is an expression that defines a change of state for a distinct feature in S.*

For example, $(f, v_1 \to v_2)$ is an atomic action term which defines a change of value for feature f in S from v_1 to v_2, where $v_1, v_2 \in V_f$. In the case when there is no change, we omit the right arrow sign; so for example, (f, v_1) means that the value of feature f in S remains v_1, where $v_1 \in V_f$.

Atomic action terms model value transition patterns for a single feature, but they do not model the association between feature values transition patterns. We augment the definition of atomic action terms to action terms by associating several transitions of feature values.

Definition 4 (Action terms) *are defined as the smallest collection of expressions for an information system S, such that:*

- *If t is an atomic action term in S, then t is an action term in S.*
- *If t_1, t_2 are action terms in S and $\wedge$ is a 2-argument functor called composition, then $t_1 \wedge t_2$ is a candidate action term in S.*
- *If t is a candidate action term in S and for any two atomic action terms $(f, v_1 \to v_2), (g, w_1 \to w_2)$ contained in t we have $f \neq g$, then t is an action term in S.*

Action terms provide an actionable knowledge; however, we still need the actions to perform in order to trigger these action terms. To do so, we use meta-actions which, when executed, trigger changes in values of some flexible features in S [2].

More formally, let us define $\mathbf{M}(S)$ as a set of meta-actions associated with an information system S. Let $f \in F$, $x \in X$, and $M \subset \mathbf{M}(S)$, then, applying the meta-actions in the set M on an object x will result in $M(f(x)) = f(y)$, where object x is converted to object y by applying all meta-actions in M to x. Similarly, $M(F(x)) = F(y)$, where $F(y) = \{f(y) : f \in F\}$ for $y \in X$, and object x is converted to object y by applying all meta-actions in M to x for all $f \in F$.

2.1 Side Effects Based on Action Terms

As stated before, the main goal of meta-actions is to trigger action rules. However, it is often the case that when applying meta-actions for the purpose of executing a specific action rule, a set of additional unrelated and potentially harmful atomic action terms is triggered. The additional action terms resulting from the meta-action application are called side effects. Meta-actions might move the values of some object's features from negative to positive (desirable positive side effects), and values of some object's features from positive to negative values (undesirable negative side effects). Even though the features transitioning from positive to negative values might result in catastrophic situations, they were not fully investigated in previous work involving action rules discovery [4]. In the following, we depict two types of side effects based on action terms and we give a brief description for each type.

Meta-actions Side Effects. Side-effects based on action terms in the context of meta-actions alone are the effects that occur for specific small clusters of objects. This type of side effects is discovered in the meta-action extraction process. It is represented by the action terms that exhibit very low or unusual likelihood of occurrence. In fact, this type of action term is very rare in our dataset, and it was extracted from a very small number of objects. We can think of this type of effects as minor effects of a meta-action that do not represent the core goal of applying this meta-action. Detecting this type of side effects is done by setting a minimum number of occurrence for the action terms (cardinal of support), or setting a minimum jump in values of cardinal of supporting set between the action terms.

Action Rules Side Effects. Side-effects based on action terms in the context of action rules are the unintended changes in the values of some flexible features that meta-actions trigger on objects. In other words, those effects are triggered by meta-actions but are outside of the intended action rule scope. To discover those side effects, we can perform two set operations. We start by performing a set difference operation between the antecedent side of the action rule and the meta-actions' action terms reported in the influence matrix. The result is then intersected with the object's precondition to get the final set of side effects.

2.2 Action Sets

The changes in flexible features, triggered by meta-actions, are commonly represented by action terms for the respective features, and reported by an influence matrix presented in [2]. However, when an information system contains multi-valued features where the same feature takes a set of values at any given object state and transitions to another set of values in a different object state, it is best to represent the transitions between the feature initial set of values and another set of values by action sets [5] that are defined as:

Definition 5 (Action Set) *An action set in an information system S is an expression that defines a change of state for a distinct feature that takes several values (multivalued feature) at any object state.*

For example, $\{f_1, f_2, f_3\} \rightarrow \{f_1, f_4\}$ is an action set that defines a change of values for feature $f \in F$ from the set $\{f_1, f_2, f_3\}$ to the set $\{f_1, f_4\}$ where $\{f_1, f_2, f_3, f_4\} \subseteq V_f$. Action sets are used to model meta-action effects for information systems with multivalued features. In addition, the usefulness of action sets is best captured by the set intersection, between the two states involved, that models neutral action sets, and set difference, between the two states involved, that models positive action sets. In the previous example, neutral and positive action sets are respectively computed as follow: $\{f_1, f_2, f_3\} \rightarrow [\{f_1, f_2, f_3\} \cap \{f_1, f_4\}]$ and $\{f_1, f_2, f_3\} \rightarrow [\{f_1, f_2, f_3\} \setminus \{f_1, f_4\}]$.

Positive and neutral action sets were previously studied in [5]. In this paper, we are mainly interested in the study of negative action sets extracted from information systems with multivalued features. These negative action sets represent negative side effects when medical meta-actions are applied, and they are best captured by the reverse set difference between the two states involved. In the previous example, negative action sets are captured as follows: $\{f_1, f_2, f_3\} \rightarrow [\{f_1, f_4\} \setminus \{f_1, f_2, f_3\}]$

3 Negative Side Effects

In healthcare, the study of side effects is mainly related to treatments and patients' conditions. In this section, we study the mining and representation of negative action sets (negative side effects) resulting from the application of meta-actions. We also show how to cluster patients based on these negative action sets and analyze the clusters.

3.1 Negative Action Sets Representation and Mining

Negative side effects are represented by action sets which model the appearance of certain diagnoses when applying a meta-action on specific patients. These diagnoses were not intended by the physician and can be harmful to the patient. The negative action sets are part of the meta-action effects, and they are best captured by the reverse set difference between the prior and posterior state of the patient. For instance, applying meta-action treatment m to patient x who is diagnosed with $F(x)_t = \{Dx_1, Dx_2, Dx_3\}$ at the prior state time t might transition the patient to a new state with the following diagnoses $F(x)_{t+1} = \{Dx_1, Dx_4\}$ at the posterior time $t+1$. This transition introduces a new diagnosis condition Dx_4 that was not present before applying m. The action set resulting is described by: $\{Dx_1, Dx_2, Dx_3\} \rightarrow [\{Dx_1, Dx_4\} \setminus \{Dx_1, Dx_2, Dx_3\}]$, where $[\{Dx_1, Dx_4\} \setminus \{Dx_1, Dx_2, Dx_3\}] = Dx_4$ represents the reverse set difference between the left hand side of the action set and its right hand side. In this example Dx_4 is seen as a negative side effect that appeared as a result of applying m to x.

Meta-actions effects extracted from information systems with multivalued features are commonly represented by an ontology that include neutral $\overline{As}$, and positive $\underline{As}$ action sets. We augment this representation by including negative action sets labeled $\overline{\underline{As}}$ and represented in red in Figure 1.

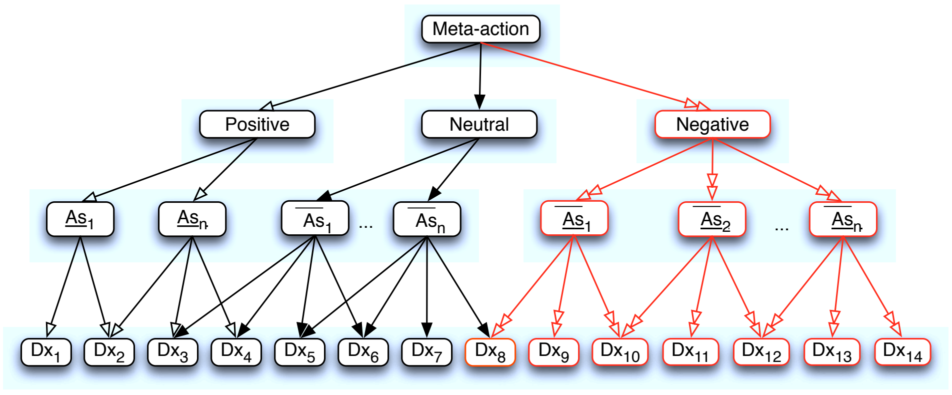


Fig. 1. Ontology representation of a meta-action with negative effects.

Once we define the format of negative action sets, we use the same action set mining technique described in [5]. In fact, we order patients by visit date, and create pairs containing two consecutive visits for each patient. The negative action sets are then extracted from those pairs for each patient and a power set is then generated to extract all possible combinations.

3.2 Patients' Clustering Based on Side Effects

Patients react similarly to some treatments that result in the same negative side effects. Therefore, it is important to keep track of the meta-actions side effects and cluster the patients who experience similar negative side effects. Clustering the patients based on the negative side-effects is done using the negative action sets. The negative action sets are supported by patients that reacted negatively to the treatments (meta-actions) applied with the respective side effects. The supporting patients for a specific negative action set constitute a cluster of patients; hence, the clustering is done by grouping those patients.

By the supporting set for a negative action set $\underline{\overline{As}} = [F(x)_{t+1} \setminus F(x)_t]$ of the form $F(x)_t \rightarrow [F(x)_{t+1} \setminus F(x)_t]$ in an information system $S = (X, F, V)$, where $F(x)_t = \{f(x)_t : f \in F\}$, we mean the set of patients $x \in X$ represented by the expression $sup(\underline{\overline{As}}) = \{x \in X : (\forall f(x) \in \underline{\overline{As}})\ [(f(x) \in F(x)_{t+1}) \wedge (f(x) \notin F(x)_t)]\ \}$. Now, $sup(\underline{\overline{As}})$ represents the set of the objects affected by the negative action set. This way each supporting set of patients represents a different cluster $sup(\underline{\overline{As_i}})$ labeled by $\underline{\overline{As_i}}$.

4 Negative Side Effects Evaluation

The negative actions sets are evaluated and analyzed using the confidence and the cardinal of supporting set -*CardSup*- that are similar to the evaluation of neutral and positive action sets. In fact, for each negative action set $\underline{\overline{As}}$, we can compute the cardinal of the support $CardSup(\underline{\overline{As}})$ as follows:

$$CardSup(\underline{\overline{As}}) = card(sup(\underline{\overline{As}})) \tag{1}$$

The $CardSup(\underline{\overline{As}})$ is a good measure of the spread or dominance of this negative side effect for a specific meta-action.

Of course, we can also compute the negative action set confidence $ActionConf(\underline{\overline{As}})$ as follows:

$$ActionConf(\underline{\overline{As}}) = \frac{CardSup(\underline{\overline{As}})}{card(\{x_i \in X :\ [\underline{\overline{As}} \subseteq F(x_i)_{t+1}]\})} \tag{2}$$

where $F(x_i)_t$ represents the initial state of the object or patient x_i. Note here that the $ActionConf(\underline{\overline{As}})$ does not model the confidence of predicting that patients with the initial state $F(x_i)_t$ will react with negative side effects to the meta-action; it rather models the confidence of the action set being a negative action set $\overline{\underline{As}}$ and not a neutral one. In other words, it does not model correlation between $F(x_i)_t$ and $\overline{\underline{As}}$.

In previous work [5], we computed the meta-action confidence $MetaConf(m)$ for m to acquire a general idea on its stability with regards to patients initial states and the positive and neutral action sets. However, we did not include the negative side effects because the purpose of the meta-action is to cure the initial diagnoses of the patients. On the other hand, the negative side effects may have originated from the correlation between the applied meta-action and some stable or unknown features and not necessarily from the initial state of the patient.

$$NegMetaConf(m) = \frac{\sum\limits_{i=1}^{n} [CardSup(\overline{\underline{As_i}}) \cdot ActionConf(\overline{\underline{As_i}})]}{\sum\limits_{i=1}^{n} CardSup(\overline{\underline{As_i}})} \tag{3}$$

where n is the number of extracted negative action sets. The negative meta-action confidence informs us about how the initial state of the patient is correlated with the negative action sets.

5 Evaluation

5.1 HCUP Dataset Description

In this paper, we used the Florida State Inpatient Databases (SID) that is part of the Healthcare Cost and Utilization Project (HCUP). The Florida SID dataset contains records from several hospitals in the Florida State. It contains over 2.5 million visit discharges from over 1.5 million patients. The dataset is composed of five tables, namely: AHAL, CHGH, GRPS, SEVERITY, and CORE. The main table used in this work is the *Core* table. The *Core* table contains over 280 features; however, many of those features are repeated with different codification schemes. In the following experiments, we used the Clinical Classifications Software (CCS) that consists of 262 diagnosis categories, and 234 procedure categories. This system is based on ICD-9-CM codes. In our experiments, we used fewer features that are described in this section. Each record in the *Core* table represents a visit discharge. A patient may have several visits in the table. One of the most important features of this table is the $VisitLink$ feature, which describes the patient's ID. Another important feature is the Key, which is the primary key of the table that identifies unique visits for the patients and links to the other tables. As mentioned earlier, a $VisitLink$ might map to multiple Key in the database. This table reports up to 31 diagnoses per discharge as it has 31 diagnosis columns. However, patients' diagnoses are stored in a random order in this table. For example, if a particular patient visits the hospital twice with heart failure, the first visit discharge may report a heart failure diagnosis at diagnosis column number 10, and the second visit discharge may report a heart failure diagnosis at diagnosis column number 22. Furthermore, it is worth mentioning that it is often the case that patients examination returns less than

Table 1. Mapping between features and concepts features.

features	**Concepts**
VisitLink	Patient Identifier
DaysToEvent	Temporal visit ordering
DXCCSn	n^{th} Diagnosis, flexible feature
PRCCSn	n^{th} Procedure, meta-action
Race, Age Range, Sex,..	Stable features
DIED	Decision Atribute

31 diagnoses. The *Core* table also contains 31 columns describing up to 31 procedures that the patient went through. Even though a patient might have gone through several procedure in a given visit, the primary procedure that occurred at the visit discharge is assumed to be the first procedure column. The *Core* table also contains an feature called *DaysToEvent*, which describes the number of days that passed between the admission to the hospital and the procedure day. This field is anonymized in order to hide the patients' identity. Furthermore, the *Core* table also contains a feature called *DIED*, that informs us on whether the patient died or survived in the hospital for a particular discharge. There are several demographic data that are reported in this table as well, such as: Race, Age Range, Sex, living area, . . . etc. Table 1 maps the features from the *Core* table to the concepts and notations used in this paper.

5.2 Experiments

We performed several experiments regarding mining negative action sets and analyzing patients' clusters based on negative side effects. We used four meta-actions to extract and analyze side effects. The four meta-actions used are referenced by the following procedure CCS codes [1]: 34, 43, 44, and 45.

We started by mining the negative action sets and evaluated them. We also give a few examples of the negative action sets mined in Table 2. For instance, coronary artery bypass graft (CCS:44) results in bacterial infection (CCS:3, ActionConf=98%). Patients were then grouped and analyzed based on side effects.

You can note from Table 3 that the *MetaConf* is smaller than the average confidence since it reflects a better global confidence of the meta-actions with regard to *CardSup*. However, Figure 2 shows that more than 73% of action sets have over 90% confidence for all meta-acions. This shows that a threshold on the *ActionConf* can be used to eliminate some negative action sets. The total number of clusters, the average *CardSup*, and the average cluster's negative action set size are also reported in Table 3 for descriptive reasons.

Table 4 shows that the number of clusters follows a Gaussian distribution [11] behavior with regard to their negative action sets sizes. In addition, the average *CardSup* decreases when the size of the cluster's action sets increases. The action set cluster with size 0 indicates no side effects; in other words, patients in this cluster did not have any side effects as a result of applying the meta-action.

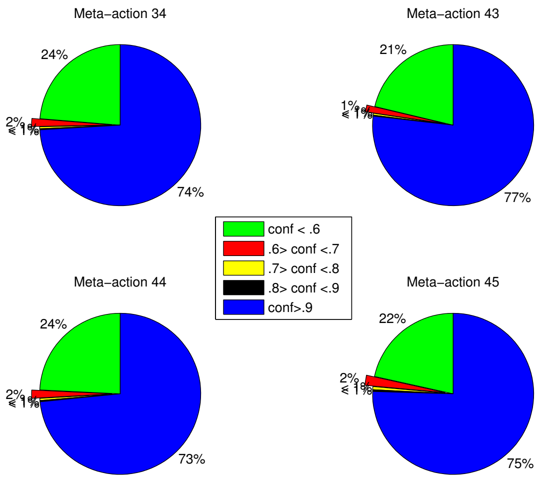


Fig. 2. Negative action sets confidence proportion.

Those tables show that increasing the size of the cluster action sets, in most of the cases, increases the average of $ActionConf$. This is due to introducing more constraints in the action sets.

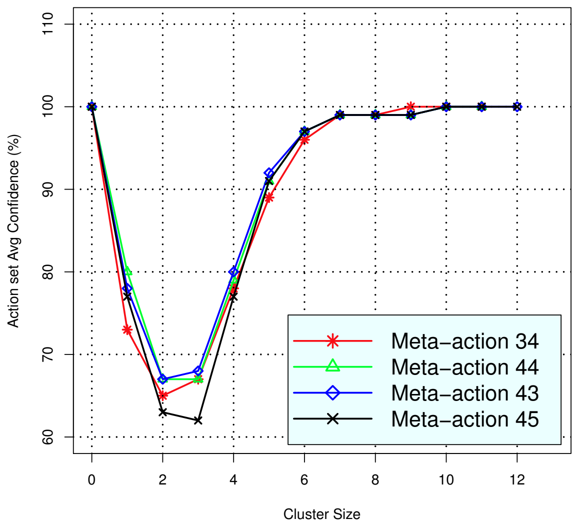


Fig. 3. Negative action sets confidence by size of clusters.

Figure 3 summarizes the trend of the action sets average confidence in a better way. This figure shows that the average action set confidence is low for action sets' clusters with sizes ranging from 1 to 4. This is due to the small number of supporting patients for these clusters.

Table 2. Examples of negative action sets for all meta-actions.

Meta-action	Negative action set	Size	CardSup	ActionConf
34	[1]	1	404	0.99
	[238]	1	308	0.83
	[134]	1	719	0.88
	[155]	1	932	0.84
	[1 , 155]	2	187	0.89
	[134 , 155]	2	366	0.77
	[134 , 1 , 155]	3	54	0.87
	[134 , 59 , 155]	3	55	0.58
43	[257]	1	429	0.93
	[254]	1	399	1.00
	[155]	1	238	0.91
	[159]	1	227	0.81
	[259 , 254]	2	102	0.89
	[257 , 254]	2	72	0.96
	[113 , 159 , 254]	3	10	1.00
	[105 , 254 , 155]	3	8	1.00
44	[254]	1	403	1.00
	[102]	1	276	0.99
	[197]	1	272	0.93
	[3]	1	214	0.98
	[2 , 244]	2	111	0.86
	[197 , 238]	2	121	0.68
	[134 , 2 , 244 , 249 , 157]	5	3	1.00
	[2 , 52 , 249]	3	12	1.00
45	[102]	1	1532	0.97
	[130]	1	495	0.91
	[153]	1	456	0.91
	[60]	1	452	0.96
	[2 , 244]	2	252	0.90
	[153 , 60]	2	127	0.95
	[146 , 120]	2	68	0.96
	[2 , 244 , 249]	3	58	0.85

Table 3. Negative clusters analysis for all meta-actions.

Meta-action	Average clusters size	Total number of clusters	Average CardSup	Average ActionConf	MetaConf
45	4.84	405931	1.69	0.84	0.53
44	4.46	178446	1.49	0.83	0.55
43	4.70	194957	1.37	0.85	0.62
34	4.63	180372	1.44	0.83	0.58

Table 4. Negative clusters analysis for meta-action 43.

Meta-action	Action set clusters size	Number of clusters	Average CardSup	Average Action-Conf
43	0	1	2645	1
	1	213	64.12	0.78
	2	6277	5.46	0.67
	3	33158	1.64	0.68
	4	54737	1.10	0.80
	5	48576	1.01	0.92
	6	30447	1.00	0.97
	7	14476	1.00	0.99
	8	5275	1.00	0.99
	9	1457	1.00	0.99
	10	296	1.00	1
	11	41	1.00	1
	12	3	1.00	1
45	0	1	12076.0000	1
	1	225	225.2400	0.77
	2	8230	13.2682	0.63
	3	59864	2.4103	0.62
	4	111169	1.2360	0.77
	5	104193	1.0493	0.91
	6	70782	1.0130	0.97
	7	35255	1.0034	0.99
	8	12466	1.0006	0.99
	9	3143	1.0000	0.99
	10	542	1.0000	1
	11	58	1.0000	1
	12	3	1.0000	1
34	0	1	2264	1
	1	213	60.11	0.73
	2	5803	5.86	0.65
	3	29814	1.86	0.67
	4	52691	1.17	0.78
	5	47620	1.02	0.89
	6	28146	1.00	0.96
	7	11794	1.00	0.99
	8	3495	1.00	0.99
	9	703	1.00	1
	10	87	1.00	1
	11	5	1.00	1
44	0	1	3905	1
	1	219	82.39	0.80
	2	6912	5.99	0.67
	3	36008	1.67	0.67
	4	54893	1.11	0.79
	5	43770	1.01	0.91
	6	23878	1.00	0.97
	7	9463	1.00	0.99
	8	2700	1.00	0.99
	9	532	1.00	0.99
	10	66	1.00	1
	11	4	1.00	1

6 Conclusion

Mining negative side-effects allows us to cluster patients with similar negative action sets. This work is very helpful for the predictability of negative side effects, and personalized action rules extraction. We have shown in this paper how negative side effects based on action terms are represented, and demonstrated how negative action sets are structured and extracted. We then presented negative action sets evaluations metrics, and analyzed patients' clusters based on these metrics for the Florida State Inpatient Databases (SID)[1]. Our results show a high confidence for the negative action sets extracted and a high negative meta-action confidence for the meta-actions examined.

References

1. Healthcare Cost and Utilization Project (HCUP). Clinical classifications software (ccs), http://www.hcup-us.ahrq.gov.
2. Ke. Wang, Y. Jiang, and A. Tuzhilin. Mining actionable patterns by role models. In *Data Engineering, ICDE '06. Proceedings of the 22nd International Conference*, pages 16–26, 2006.
3. Z.W. Raś and A. Wieczorkowska. Action-rules: How to increase profit of a company. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 587–592, London, UK, 2000. Springer.
4. H. Touati, J. Kuang, A. Hajja, and Z.W. Ras. Personalized action rules for side effects object grouping. *the Special Issue on "Knowledge Discovery", G. Wang (Ed.) , International Journal of Intelligence Science (IJIS)*, 3(1A):24–33, 2013.
5. H. Touati, Z.W. Ras, J. Studnicki, and A. Wieczorkowska. Mining surgical meta-actions effects with variable diagnoses number. In *Proceedings of ISMIS 2014 in Roskilde, Denmark*, volume 8502, pages 254–263. Springer, 2014.
6. K.S. Kolibaba, J.A. Sterchele, AD. Joshi, M. Forsyth, E. Alwon, H. Beygi, and G.T. Kennealey. Demographics, treatment patterns, safety, and real-world effectiveness in patients aged 70 years and over with chronic lymphocytic leukemia receiving bendamustine with or without rituximab: a retrospective study. *Therapeutic Advances in Hematology*, 4(3):157171, 2013.
7. F. C. Allen-Ramey, S. Gupta, and M.D. DiBonaventura. Patient characteristics, treatment patterns, and health outcomes among copd phenotypes, 2012.
8. P. Lorigan, M. Marples, M. Harries, J. Wagstaff, A.G. Dalgleish, R. Osborne, A. Maraveyas, S. Nicholson, N. Davidson, Q. Wang, L. Pericleous, U. Bapat, and M.R. Middleton. Treatment patterns, outcomes, and resource utilization of patients with metastatic melanoma in the u.k. *British Journal of Dermatology*, page 8795, 2014.
9. M. Thangamani, P. Thangaraj, and Bannari. Automatic medical disease treatment system using datamining. In *Information Communication and Embedded Systems (ICICES), 2013 International Conference on*, pages 120–125, Feb 2013.
10. Z. Pawlak. Information systems - theoretical foundations. *Information Systems Journal*, 6:205–218, 1981.
11. NR Goodman. Statistical analysis based on a certain multivariate complex gaussian distribution (an introduction). *Annals of mathematical statistics*, pages 152–177, 1963.

Pitch-Related Identification of Instruments in Classical Music Recordings

Elżbieta Kubera[1] and Alicja A. Wieczorkowska[2]

[1] University of Life Sciences in Lublin, Akademicka 13, 20-950 Lublin, Poland
elzbieta.kubera@up.lublin.pl
[2] Polish-Japanese Institute of Information Technology,
Koszykowa 86, 02-008 Warsaw, Poland
alicja@poljap.edu.pl

Abstract. Identification of particular voices in polyphonic and polytimbral music is a task often performed by musicians in their everyday life. However, the automation of this task is very challenging, because of high complexity of audio data. Usually additional information is supplied, and the results are far from satisfactory. In this paper, we focus on classical music recordings, without requiring the user to submit additional information. Our goal is to identify musical instruments playing in short audio frames of polyphonic recordings of classical music. Additionally, we extract pitches (or pitch ranges) which combined with instrument information can be used in score-following and audio alignment, see e.g. [8], [17], or in works towards automatic score extraction, which are a motivation behind this work. Also, since instrument timbre changes with pitch, separate classifiers are trained for very narrow pitch ranges for each instrument. Four instruments are investigated, representing stringed and wind instruments. Random forests are applied as a classification tool, and the results are presented and discussed in the paper.

1 Introduction

Music Information Retrieval (MIR) is an area of interest not only for musicians, but for virtually everybody who listens to music and has access to any music collection. For example, one can look for a piece of music on the basis of a tune hummed or sung, through so called query-by-humming [16], or through query by example, i.e. audio query, even using mobile devices [23], [26]. Musicians may have more sophisticated needs, including identification of played notes in audio files, and assigning these notes to particular voices (instruments). The goal of our research is to extract information about instruments playing in polyphonic recordings of classical music, and combine it with pitch information (pitch describes the degree of highness or lowness of a tone [21], i.e. how high or low the tone is).

Both instrument identification and multi-pitch tracking are research targets in MIR community. Multi-pitch estimation has been performed through non-negative matrix factorization [1], or Bayesian non-negative harmonic-temporal factorization [22]; it can also be performed using Gaussian Mixture Models (GMM) originating from the speech domain [9], and other methods. The final goal is identification of as much of the score as possible, towards automatic music transcription [10], [27] but usually additional information must be supplied together with the input audio data. Research on automatic instrument identification has also been performed, using various approaches [6], [5], [12], [14].

In our paper, we focus on identification of musical instruments in music recordings, which is continuation of our previous research [12]. However, previously we did not extract pitch information. This time we extract information about pitch or pitches played in a given audio segment, and use it for instrument recognition. Random forests are applied as a classification tool, and classical music recordings are used as audio data, with 4 target instruments investigated in their full scale.

2 Audio Data

Classical music is usually played with typical instrument sets, and therefore these instruments were investigated in our research. Since preparing ground-truth data for testing is a tedious task, we decided to perform our tests on two pieces of music, taken from RWC Classical Music collection [3], and use the first minutes of the selected recordings. These pieces of music are:

- No. 18 (C18), J. Brahms, Horn Trio in Eb major, op.40. 2nd mvmt.; instruments playing in the first minute: piano, French horn, violin;
- No. 44 (C44), N. Rimsky-Korsakov, The Flight of the Bumble Bee; flute and piano.

The target instruments we want to identify in these recordings are flute, piano, French horn, and violin.

The sounds for training classifiers were taken from RWC [4], MUMS [20], and IOWA [25] audio data. The data were basically in stereo format, and the mix of both channels was used in our experiments.

2.1 Parametrization

The audio data were parameterized as a preprocessing, which means that a sequence of samples representing amplitude changes in time was replaced with a much shorter sequence of numbers, i.e. parameters describing various sound properties. A feature vector of 58 features was extracted for each analyzed frame, as we did in our previous research [12]; most of them represent MPEG-7 low-level audio descriptors, often used in audio research [7]. These features are extracted for 120-ms audio frames, analyzed using Fourier transform with Hamming windowing. Our audio data were recorded with 16-bit resolution and 44.1 kHz sampling rate, so 120 ms corresponds to 5292 samples; these samples were zeropadded (i.e. with added zeros) to 8192 samples, in order to apply FFT (Fast

Fourier Transform), which requires the number of samples being the power of two. The features used in our work are listed below [12]):

– *Audio Spectrum Flatness*, $\mathit{flat}_1, \ldots, \mathit{flat}_{25}$ — 25 parameters representing the flatness of the power spectrum within a frequency bin for selected bins; 25 out of 32 frequency bands were used;
– *Audio Spectrum Centroid* — the power weighted average of the frequency bins in the power spectrum. Coefficients were scaled to an octave scale anchored at 1 kHz [7];
– *Audio Spectrum Spread* — RMS (root mean square) of the deviation of the log frequency power spectrum wrt. *Audio Spectrum Centroid* [7];
– *Energy* — energy (in log scale) of the spectrum;
– *MFCC* — 13 mel frequency cepstral coefficients. The cepstrum was calculated as the logarithm of the magnitude of the spectral coefficients, and then transformed to the mel scale, reflecting properties of the human perception of frequency. 24 mel filters were applied, and the results were transformed to 12 coefficients. The 13^{th} parameter is the 0-order coefficient of MFCC, corresponding to the logarithm of the energy [19];
– *Zero Crossing Rate* of the time-domain representation of the sound wave; a zero-crossing is a point where the sign of the function changes;
– *Roll Off* — the frequency below which 85% (experimentally chosen threshold) of the accumulated magnitudes of the spectrum is concentrated;
– *NonMPEG7 - Audio Spectrum Centroid* — the linear scale version of *Audio Spectrum Centroid*;
– *NonMPEG7 - Audio Spectrum Spread* — the linear scale version of *Audio Spectrum Spread*;
– *Flux* – the sum of squared differences between the magnitudes of the DFT (Discrete Fourier Transform) points calculated for the current frame and the previous one. In the case of the first frame, flux is equal to zero by definition.
– *Chroma* - 12-element chroma vector [18] of summed energy of pitch classes, corresponding to the equal-tempered scale, i.e. C, C#, D, D#, E, F, F#, G, G#, A, A#, and B. A pitch class consists of pitches of the same name through all octaves. Chroma vector was calculated using Chroma Toolbox [15]; the sampling rate was converted to 22.05 kHz to apply this toolbox.

3 Classification with Random Forests

Random forest (RF) classifiers have been applied as classification tool, since their proved successful in our previous research [12]. RF is a set of decision trees, constructed using procedure minimizing bias and correlations between individual trees. Each tree is built using a different N-element bootstrap sample of the N-element training set, i.e. obtained through drawing with replacement from the original N-element set. About 1/3 of the training data are not used in the bootstrap sample for any given tree. For a K-element feature vector representing objects, k attributes (features) are randomly selected ($k \ll K$, often $k = \sqrt{K}$)

at each stage of tree building, i.e. for each node of any particular tree in RF. The best split on these k attributes is used to split the data in the tree node, and Gini impurity criterion is applied (minimized) to choose the split. The Gini criterion is the measure of how often an element would be incorrectly labeled if labeled randomly, according to the distribution of labels in the subset. Each tree is grown without pruning to the largest possible extent. A set of M trees is obtained through repeating this randomized procedure M times. Classification of each object is made by simple voting of all trees in such a random forest [2].

In the described research we decided to train a separate RF for instrument and a narrow pitch range (the main pitch and the two neighboring semitones) combination. The analyzed frame is 120-ms long, so the obtained spectral resolution allows determining even the lowest sounds of the investigated instruments. A general scheme of building the classification model in the described work is shown in Figure 1.

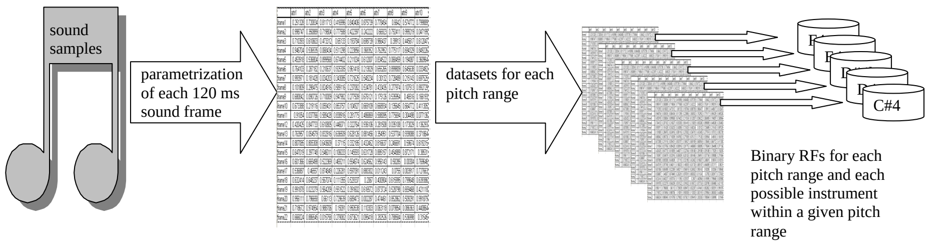

sound samples
parametrization of each 120 ms sound frame
datasets for each pitch range
C#4
Binary RFs for each pitch range and each possible instrument within a given pitch range

Fig. 1. Building the classification model in the described work

3.1 Instrument and Pitch Identification

At the preprocessing stage, the average RMS of the whole analyzed audio piece was calculated. Frames of RMS below 1/4 of this level were considered to represent silence and ignored in the next stages; namely, pitch information is not extracted for these frames, and for each instrument the probability of this instrument playing in this frame is set as equal to zero (classifiers are not applied). DFT was calculated for the remaining frames, and spectral peaks were found in 4096-point halves of log-amplitude 8192-point spectrums. The investigated music pieces were analyzed frame by frame, with 40 ms hop size.

Firstly, a global maximum of the amplitude spectrum was found. Secondly, 8-point subsegments were analyzed (with 2-point hop size) in order to find a maximum in each subsegment, being also a local maximum (i.e. surrounded by values lower than the maximal value) [28]. If the maximal value was at the border of the subsegment, the neighboring segment was also used, to find maximums at the borders as well. Next, the extracted maximum was put in the candidate list, if it was not added to the list before, and only if its value exceeded the minimum spectrum value extended by 3/4 of the difference between the maximum and minimum value. Weights were assigned to each peak as follows: for a candidate peak, log amplitudes of 10 consequent multiples of its frequency (i.e. 10 potential harmonics) are summed up. Additionally, neighboring spectral bins are also included in searching of potential harmonics. After the list of weights for candidate peaks is completed, it is sorted with descending order of weights, and the biggest gap between weights is found. Finally, peaks with weights above this cut-off threshold are kept, and they represent pitches found for the analyzed audio frame. If the neighboring frames contain a given pitch, it is also added. Also, if this pitch is present in a given frame and is not in the neighboring ones, then the next neighboring frame to the right is also checked, and if this pitch is indicated here, it remains on the pitch list, otherwise it is removed.

Instrument timbre changes with pitch [11], and spectral peaks corresponding to the pitch may spread through neighboring spectral bins, where a spectral bin represents the frequency range equal to 1/frame_length of the sampling rate. The lowest analyzed pitches, i.e. from A0 to G#1 (in MIDI notation) represent piano sounds; pitches from A1 to F#3 may represent piano or French horn. Therefore, if pitch A1 or lower is recognized, then piano is indicated as an instrument playing this sound, without using classifiers. For sounds between A1 and F#3, classifiers for piano and French horn are applied. For higher pitches, classifiers for other target instruments are applied, too. Additionally, these classifiers are separately build for pitch ranges corresponding to the recognized pitch, but comprising both neighboring semitones, i.e. below and above the recognized pitch. Since the lowest pitch for the investigated instruments is A0, spectral peaks below A0 were ignored in the described work.

Our classifiers are trained to recognize an instrument sound of a given pitch range. A separate binary random forest is calculated for each instrument-pitch range pair. Altogether 208 RF classifiers were trained, each one aiming at the recognition of whether the target instrument is playing the pitch labeling this RF. For every spectral peak listed on the pitch list found in the spectrum, the classifiers for instruments encompassing this frequency are applied. In the case of the lowest and highest frequencies, piano is automatically given as output, because no other instruments produce these sounds. For every detected pitch, we indicate all instruments that play in the given frame, according to probabilities yielded by RFs (each RF gives as the output the probability of a target instrument playing the sound of a given pitch in the frame). We take maximum of these probabilities through all pitches detected in a given frame to obtain the probability of each instrument playing in this frame. If all probabilities are below 20%, no instrument is indicated at the output; if all exceed 80%, all are indicated at the output. Otherwise, the biggest differences between neighboring probabilities is found and it constitutes cut-off threshold on the instrument list.

3.2 Cleaning

The output of the classifiers is frame-based. The obtained results are then cleaned. Namely, if the neighboring frames contain a given instrument, it is also added, with the probability being the average of its neighbors. Also, if an instrument is present in a given frame and is not in the neighboring ones, then the next neighboring frame to the right is also checked, and if the instrument is indicated here, it remains on the list, otherwise it is removed.

The final output is given for 0.5-second segments, with ground-truth carefully manually labeled. Again, average probability for each instrument through all frames in this segment is calculated. If all probabilities are below 10%, no instrument is indicated at the output; if all exceed 90%, all are indicated at the output. The instrument list is cut off where the biggest probability drop is. A general scheme of predicting instrument for test audio samples is shown in Figure 2.

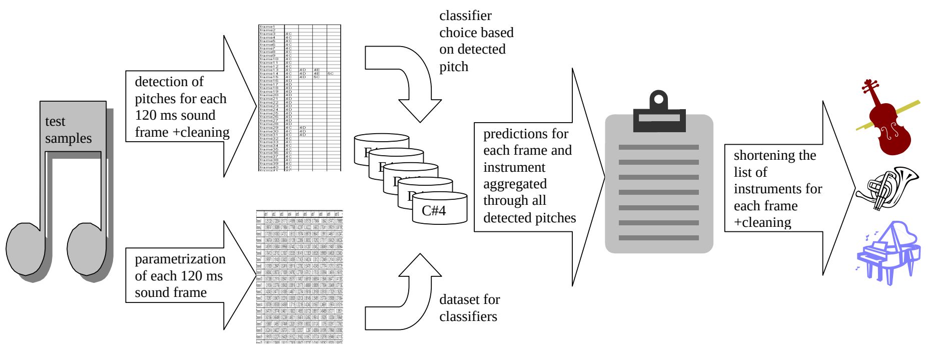

test samples
detection of pitches for each 120 ms sound frame +cleaning
parametrization of each 120 ms sound frame
classifier choice based on detected pitch
C#4
dataset for classifiers
predictions for each frame and instrument aggregated through all detected pitches
shortening the list of instruments for each frame +cleaning

Fig. 2. Predicting instruments for test samples in the described work

3.3 Training of Random Forests

Training of RFs was performed on 120 ms sound frames, taken from audio recordings with 40 ms overlap between the frames. The training set of no more than 40,000 frames was based on single sounds of musical instruments, taken from RWC [4], MUMS [20], and IOWA [25] sets of single sounds of musical instruments, and on mixes of three instrument sounds. Positive examples for a target instrument were represented by single sounds and mixes containing the target instrument, and negative examples were represented by single sounds of other instruments or mixes without the target instrument. The set of instruments in mixes was always typical for classical music, with the probability of instruments playing together in the mix reflecting the probability of these instruments playing together in the RWC Classical Music Database.

4 Results

The outcomes of our experiments are presented using true positives (TP - the number of instrument correctly identified by the classification system for a given sound segment), true negatives (TN - the number of instruments with correct negative answer of the classification system), false positives (FP - the number of instruments with positive answer of the classification system, but actually not playing) and false negatives (FN - the number of instruments with negative answer of the classification system, but actually playing). The following measures are used [13]:

– precision pr calculated as [24]:

$$pr = \frac{TP + 1}{TP + FP + 1},$$

– recall rec calculated as [24]:

$$rec = \frac{TP + 1}{TP + FN + 1},$$

– f-measure f_{meas} calculated as:

$$f_{meas} = \frac{2 \cdot pr \cdot rec}{pr + rec},$$

– accuracy acc calculated as:

$$acc = \frac{TP + TN}{TP + TN + FP + FN}.$$

As mentioned before, we decided to use RFs because they outperformed other classifiers in research on musical instrument sounds. For illustration purposes, the results of instrument identification using a set of binary RFs and a set of binary k-Nearest Neighbor (k-NN) classifiers are shown in Table 1; each binary classifier was trained to identify whether a target instrument is playing or not, without pitch identification. The results show superiority of RFs over k-NN. Additional drawback of k-NN classifier is that it is very slow in this case, i.e. for 4 classes (instruments).

Table 1. Results of the recognition of musical instruments for C18 from RWC Classical recordings for RFs and k-NN

Result	RFs	k-NN
precision	95%	85%
recall	68%	49%
f-measure	77%	61%
accuracy	66%	46%

The results of identification of instruments based on pitch extraction for pieces no. 18 and 44 from RWC Classical Music are shown in Table 2 and Table 3 respectively. As we can see, precision for C18 is high, and in the case of C44 it is low. Also, the recall in C18 for French horn is very low.

The details of identification for 0.5-second segments of each piece are shown in Figures 3 and 4. These figures illustrate instruments only, without identifying pitches, as this would require very tedious labeling of ground truth data, which was already an arduous task. As we can see, flute was only once indicated (falsely) in C18, whereas French horn was hardly ever indicated in this piece (numerous false negatives). In the case of C44, violin was often incorrectly indicated (numerous false positives). However, we have to mention here that this is an extremely difficult piece to analyze, as it is very fast, and it would probably require a shorter analyzing frame.

Table 2. Results of the pitch&instrument-based recognition of musical instruments for C18 from RWC Classical recordings. Instruments playing in this piece: French horn, piano, and violin

Result	*flute*	*French horn*	*piano*	*violin*	Average
TP	0	2	81	81	
FP	1	0	0	18	
FN	0	97	38	9	
TN	119	21	1	12	
precision	50%	100%	100%	82%	83%
recall	100%	3%	68%	90%	65%
f-measure	67%	6%	81%	86%	60%
accuracy	99%	19%	68%	78%	66%

Table 3. Results of the pitch&instrument-based recognition of musical instruments for C44 from RWC Classical recordings. Instruments playing in this piece: flute and piano

Result	*flute*	*French horn*	*piano*	*violin*	Average
TP	47	0	17	0	
FP	13	1	42	116	
FN	54	0	30	0	
TN	6	119	31	4	
precision	79%	50%	30%	1%	40%
recall	47%	100%	38%	100%	71%
f-measure	59%	67%	33%	2%	40%
accuracy	44%	99%	40%	3%	47%

We also performed tests on mixes, in order to analyze the quality of pitch recognition, as in the case of mixes we have immediate access to ground truth labeling. The polyphony level tested was 2-4 sounds. The precision of pitch identification was 83%. The recall decreased with the polyphony level, and for 2 sounds it was equal to 55%, for 3 sounds – 43%, and for 4 sounds – 35%.

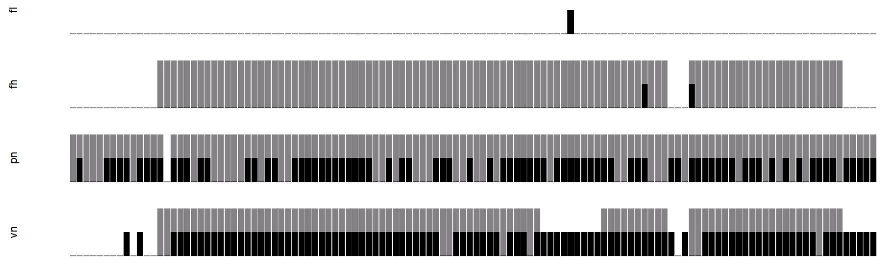


Fig. 3. Ground truth vs. identified data for RWC Classic piece no. 18. Ground truth is marked in gray, and the identified instruments are marked in black. Horizontal axis represents time (time unit equal to 0.5 s) for the first minute of C18

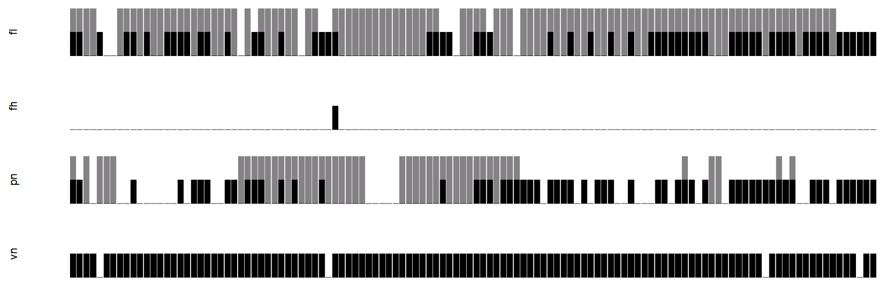


Fig. 4. Ground truth vs. identified data for RWC Classic piece no. 44. Ground truth is marked in gray, and the identified instruments are marked in black. Horizontal axis represents time (time unit equal to 0.5) for the first minute of C44

However, 9% of the errors were actually octave errors (correct pitch name, one octave higher), which are quite a common mistake in pitch tracking, but such errors are considered to be less significant than other errors – for instance, in solfeggio, it is allowed to read pitches in a different octave, more convenient for a reader.

Since the results for the presented classification methodology are rather moderate, we decided to perform experiments also for classifiers dedicated to instruments only, without constructing separate classifiers for each pitch-instrument pair, i.e. for 4 classifiers (corresponding to instruments) only, instead of 208, as it is supposed that such a big number of narrow-target (with little training data) classifiers may deteriorate the results. The outcome for such 4 RF classifiers are presented in Table 4 for C18. As we can see, the quality of results is similar, although differs for particular instrument. For instance, precision for violin is higher in this case, and recall is lower, but for French horn precision is higher for pitch&instrument based classification.

Table 4. Results of the instrument-based recognition of musical instruments for C18 from RWC Classical recordings. Instruments playing in this piece: French horn, piano, and violin

Result	*flute*	*French horn*	*piano*	*violin*	Average
TP	0	4	81	75	
FP	0	2	1	14	
FN	0	95	38	15	
TN	120	19	0	16	
precision	100%	71%	99%	84%	89%
recall	100%	5%	68%	84%	64%
f-measure	100%	9%	81%	84%	69%
accuracy	100%	19%	68%	76%	66%

5 Summary and Conclusions

In this paper we investigated automatic recognition of instrument for the played pitch (or pitch range) for selected 4 instruments typical for classical music. All instruments produced sounds of definite pitch. This paper is an extension to our previous study, where we recognized musical instruments. In this paper we were aiming at identifying pitch or pitch range as well, using a combined pitch and instrument recognition approach. Two pieces from RWC Classical Music database were used for testing, namely No. 18 and No. 44. The results are presented with indicating instrument only, to avoid laborious preparing of ground truth data. The results show many false positives for violin in the case of No. 44, but correct indication that flute does not play in No. 18, with exception of one 0.5-second segment. Also, since we extract pitch information, we can apply harmonic features as well, where multiples of the fundamental frequency are analyzed, and the enriched feature set may allow more precise classification. This is planned in our future research. Finally, after improving the results, we would like to identify as many instruments as possible playing in each analyzed frame, and improve data cleaning procedures.

Acknowledgments. This work was partially supported by the Research Center of PJIIT, supported by the Ministry of Science and Higher Education in Poland.

References

1. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: Discriminative Non-negative Matrix Factorization for Multiple Pitch Estimation. In: 13th International Society for Music Information Retrieval Conference (ISMIR), pp. 205–210 (2012)
2. Breiman, L.: Random Forests. Machine Learning 45, 5–32 (2001)
3. Goto, M., Hashiguchi, H., Nishimura, T., Oka, R.: RWC Music Database: Popular, Classical, and Jazz Music Databases. In: Proceedings of the 3rd International Conference on Music Information Retrieval, pp. 287–288 (2002)
4. Goto, M., Hashiguchi, H., Nishimura, T., Oka, R.: RWC Music Database: Music Genre Database and Musical Instrument Sound Database. In: 4th International Conference on Music Information Retrieval, pp. 229–230 (2003)
5. Heittola, T., Klapuri, A., Virtanen, A.: Musical Instrument Recognition in Polyphonic Audio Using Source-Filter Model for Sound Separation. In: 10th Int. Society for Music Information Retrieval Conf. (2009)
6. Herrera-Boyer, P., Klapuri, A., Davy, M.: Automatic Classification of Pitched Musical Instrument Sounds. In: Klapuri, A., Davy, M. (eds.) Signal Processing Methods for Music Transcription. Springer Science+Business Media LLC (2006)
7. ISO: MPEG-7 Overview, `http://www.chiariglione.org/mpeg/`
8. Izmirli, O., Sharma, G.: Bridging Printed Music and Audio Through Alignment Using a Mid-level Score Representation. In: 13th International Society for Music Information Retrieval Conference (ISMIR), pp. 61–66 (2012)
9. Kameoka, H., Nishimoto, T., Sagayama, S.: Multi-pitch Detection Algorithm Using Constrained Gaussian Mixture Model and Information Criterion for Simultaneous Speech (2004)
10. Kirchhoff, H., Dixon, S., Klapuri, A.: Multi-Template Shift-Variant Non-Negative Matrix Deconvolution for Semi-Automatic Music Transcription. In: 13th International Society for Music Information Retrieval Conference (ISMIR), pp. 415–420 (2012)
11. Kitahara, T., Goto, M., Okuno, H.G.: Musical Instrument Identification Based on F0-Dependent Multivariate Normal Distribution, In: ICME, pp. 409–412 (2003)
12. Kubera, E. Wieczorkowska, A.: Mining Audio Data for Multiple Instrument Recognition in Classical Music. In: New Frontiers in Mining Complex Patterns NFMCP 2013, International Workshop, held at ECML-PKDD 2013 (2013)
13. Kubera, E., Wieczorkowska, A., Skrzypiec, M.: Influence of Feature Sets on Precision, Recall, and Accuracy of Identification of Musical Instruments in Audio Recordings. In: ISMIS 2014, Proceedings.
14. Martins, L.G., Burred, J.J., Tzanetakis, G., Lagrange, M.: Polyphonic instrument recognition using spectral clustering. 8th International Society for Music Information Retrieval Conference (ISMIR) (2007)

15. Max-Planck-Institut Informatik: Chroma Toolbox: Pitch, Chroma, CENS, CRP, `http://www.mpi-inf.mpg.de/resources/MIR/chromatoolbox/`
16. MIDOMI: Search for Music Using Your Voice by Singing or Humming, `http://www.midomi.com/`
17. Miotto, R., Montecchio, N., Orio, N.: Statistical Music Modeling aimed at Identification and Alignment. In: Ras, Z.W., Wieczorkowska, A.A. (eds.): Advances in Music Information Retrieval. Studies in Computational Intelligence, Vol. 274, Springer Heidelberg, pp. 189–214 (2010)
18. Müller, M.: Information Retrieval for Music and Motion. Springer, Heidelberg (2007)
19. Niewiadomy, D., Pelikant, A.: Implementation of MFCC vector generation in classification context. J. Applied Computer Science, Vol. 16, No. 2, pp. 55–65 (2008)
20. Opolko, F., Wapnick, J.: MUMS — McGill University Master Samples. CD's (1987)
21. Oxford University Press: Oxford Dictionaries, `http://www.oxforddictionaries.com/`
22. Sakaue, D., Otsuka, T., Itoyama, K., Okuno, H.G.: Bayesian Nonnegative Harmonic-Temporal Factorization and Its Application to Multipitch Analysis. In: 13th International Society for Music Information Retrieval Conference (ISMIR), pp. 91–96 (2012)
23. Shazam Entertainment Ltd `http://www.shazam.com/`
24. Subrahmanian, V.S.: Principles of Multimedia Database Systems. Morgan Kaufmann, San Francisco (1998)
25. The University of IOWA Electronic Music Studios: Musical Instrument Samples, `http://theremin.music.uiowa.edu/MIS.html`
26. TrackID, `https://play.google.com/store/apps/details?id=com.sonyericsson.trackid`
27. Vincent, E., Rodet, X.: Music transcription with ISA and HMM. In: 5th International Conference on Independent Component Analysis and Blind Signal Separation (ICA) pp. 1197–1204 (2004)
28. Zhang, X., Marasek, K., Ras, Z.W.: Maximum Likelihood Study for Sound Pattern Separation and Recognition. In: Proceedings of the IEEE CS International Conference on Multimedia and Ubiquitous Engineering (MUE 2007), Seoul, Korea, pp. 807–812 (2007)

Parallel Multicut Segmentation via Dual Decomposition

Julian Yarkony[1 3], Thorsten Beier[1], Pierre Baldi[3], and Fred A. Hamprecht[1]

[1] University of Heidelberg
Heidelberg Collaboratory for Image Processing (HCI)
Heidelberg Germany
[2] University of California Irvine, Department of Computer Science
Irvine California

Abstract. We propose a new outer relaxation of the multicut polytope, along with a dual decomposition approach for correlation clustering and multicut segmentation, for general graphs. Each subproblem is a minimum st-cut problem and can thus be solved efficiently. An optimal reparameterization is found using subgradients and affords a new characterization of the trivial LP relaxation of the multicut problem, as well as informed decoding heuristics. The algorithm we propose for solving the problem distributes the computation and is amenable to a parallel implementation.

Keywords: Dual Decomposition, Multi-Cut Segmentation, Correlation Clustering

1 Introduction

We study the *multicut problem* (7), an optimization problem over the set of all segmentations of a finite graph. A segmentation of a graph is a partition of the node set into connected subsets. For complete graphs, the multicut problem specializes to an optimization problem over the set of all partitions of the node set (because every subset of nodes is connected).

The multicut problem is fundamental to machine learning in the guises of correlation clustering (8; 5) and image segmentation (22; 1; 2; 3; 12). More generally, it addresses the maximum-a-posteriori (MAP) inference problem for a broad class of discrete Markov random fields, namely all second-order graphical models which are invariant under all global permutations of labels (20).

The multicut problem is interesting also from a purely theoretical perspective because the set of all segmentations of a graph is non-trivial. The geometry of this set (that is, the geometry of the cut polytope, cf. Section 3) has been examined by (6; 7; 9) and in great detail by (10). Since optimization over the set of all segmentations is known to be NP-hard, a complete characterization of the cut polytope in terms of its facets is, however, unlikely. In practice, one therefore optimizes not explicitly over the cut polytope but, typically, over an outer relaxation which is tightened recursively by cutting planes (11). A brief overview of such approaches is given in Section 2.

In this work, starting from the trivial linear programming (LP) relaxation of the multicut problem in Section 3, we propose the following contributions:

Firstly, we offer an *instance-dependent relaxation of the multicut problem* which is more parsimonious than even the trivial LP relaxation while affording the same bound (Section 4). We establish the equality of bounds by showing, for a subset of constraints that depend on the problem instance, that these constraints, despite defining facets of the cut polytope, cannot become active at solutions. The new relaxation motivates a Lagrangian decomposition of the segmentation problem whose subproblems are induced neither by small cliques, as in (21), nor by planar graphs, as in (22).

Secondly, *we show, for the dual of the Lagrangian decomposition, that subproblems can be cast as optimization problems over sets of 2-colorable segmentations* (Section 5.1). Their solution can be reduced to the minimum st-cut problem, which we do, and can thus be found more efficiently than the solution of the trivial LP relaxation. As with any dual decomposition, the computation is split between two tasks: on the one hand, solving the subproblems, which can be done independently and in parallel; and on the other hand, optimizing the reparameterization which we achieve using subgradients (Section 6), and show how to achieve using message passing (appendix).

Thirdly, we suggest a *new and efficient rounding heuristic* (Section 6.1) for mapping solutions of the relaxed problem to feasible segmentations.

Combining these contributions, we define the first distributed algorithm for solving the multicut problem approximately, with guaranteed and well-understood lower and upper bounds. We compare our algorithm, which we call dual multicut, to existing work on problem instances from (22) and synthetic problems.

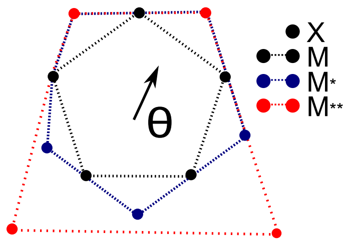


Fig. 1. Schematic of the multicut polytope and its relaxations studied here. Our outer relaxation $M^{**}(G)$ is defined by fewer inequalities, but is just as tight as the cycle polytope $M^{*}(G)$ where it matters, around the vertex that is optimal for an instance characterized by edge weights θ.

2 Related Work

The state of the art in solving instances of the NP-hard multicut problem in practice is to first solve a linear programming outer polytope relaxation. Several hierarchies of outer relaxations of the cut polytope are known, general (16) and specific (10). Of practical interest, thanks to efficient separation procedures (6; 7), are the trivial LP relaxation, that is, the intersection of the half-spaces defined by all facet-defining cycle inequalities (6), as well as the tightening of this relaxation by odd-wheel inequalities (7).

The second step is to then either map the solution of the relaxed problem to a feasible segmentation using an efficient heuristic (in polynomial time), or further tighten the relaxation using general branch-and-bound or branch-and-cut techniques, until the solution of the LP becomes integral and the problem has been solved to optimality (in exponential time in the worst case).

A different approach has been suggested by (22) for the special case of planar graphs. Their column generating algorithm affords heuristic feasible solutions at any time, thus providing approximate solutions early, as can be seen from our experiments with planar graphs in Section 7. However, it is restricted to planar graphs, unlike the algorithm we propose, which is general.

3 Segmentations and Multicuts

This section summarizes salient definitions and results, most of which are due to (6, 7, 9).

A segmentation of a weighted graph $G = (V, E)$ is a partition of the node set into connected subsets (segments). Given edge weights $\theta \in \mathbb{R}^E$, the weight of a segmentation is defined as the sum of weights of those edges that connect different segments. We refer to these weights as potentials.

One way of encoding a segmentation is in terms of a node labeling $l : V \to \{1, \dots, |V|\}$ such that (i) within each segment, all nodes have the same label, and (ii) the labels of any two adjacent segments are distinct.

A different encoding and, in fact, a characterization of a segmentation is the set of edges that straddle different segments. Not every subset of edges defines a segmentation. Those subsets of edges that do define segmentations are known as the *multicuts* of the graph. They are characterized by the indicator vectors $x \in \{0, 1\}^E$ such that

$$\forall c \in C(G) \quad \forall f \in c \quad x_f \leq \sum_{e \in c \setminus \{f\}} x_e \ . \tag{1}$$

Here, $C(G)$ denotes the set of all cycles in G. We denote by $X(G)$ the set of indicator vectors of all multicuts. We refer to an edge $e \in E$ as being *cut* if $x_e = 1$, and *uncut* if $x_e = 0$. The famed *cycle inequalities* (1) guarantee that no cut edge can separate nodes that are part of the same connected component.

The *multicut problem* consists in finding a multicut (and thus, a segmentation) with minimum weight

$$D := \min_{x \in X(G)} \sum_{e \in E} \theta_e \, x_e \tag{2}$$

This discrete problem is formulated equivalently as a linear program over the multicut polytope $M(G) := \text{conv}(X(G))$, that is, over the convex hull of $X(G)$, as

$$D = \min_{x \in M(G)} \sum_{e \in E} \theta_e \, x_e \tag{3}$$

While the system of inequalities defining $M(G)$ is believed to be exponentially large, several non-trivial outer relaxations of polynomial size are known. One of these is the *cycle polytope* $M^*(G) := \{x \in [0,1]^E \mid (1)\}$, that is, the intersection of the half-spaces defined by all cycle inequalities. The cycle polytope contains vertices that do not correspond to a convex combination of segmentations. However, it contains no additional integer vertices (7). Therefore, $X(G) = M^*(G) \cap \{0,1\}^E$. The problem

$$D^* = \min_{x \in M^*(G)} \sum_{e \in E} \theta_e \, x_e \tag{4}$$

is known as the *trivial LP relaxation* of the multicut problem. Its solution establishes a lower bound $D^* \leq D$.

4 Outer Relaxation of the Cycle Polytope

We now define an outer relaxation $M^{**}(G) \supseteq M^*(G)$ of the cycle polytope $M^*(G)$, by dropping from its definition all cycle inequalities for which the pivot edge f has non-negative potential, and show that optimization over this less complex polytope affords the same bound as optimization over the cycle polytope. This statement is formalized in

Lemma 1. *For any graph $G = (V, E)$ and edge weights $\theta \in \mathbb{R}^E$, let $M^{**}(G)$ denote the set of all $x \in [0,1]^E$ such that*

$$\forall c \in C(G) \quad \forall f \in c \mid \theta_f < 0 \quad x_f \leq \sum_{e \in c \setminus \{f\}} x_e \ . \tag{5}$$

Then,

$$D^* = \min_{x \in M^{**}(G)} \sum_{e \in E} \theta_e \, x_e \ . \tag{6}$$

See Appendix for the proof.

5 Lagrangian Decomposition

We now define a decomposition of problem (6) into subproblems. Each subproblem, indexed by $n \in N$, is defined with respect to an auxiliary graph G^n that has the same nodes and edges as the original graph G but, possibly, a different potential vector. For every edge $e \in E$, the total potential θ_e is distributed among the subproblems. Specifically:

Definition 1: For any graph $G = (V, E)$, any potentials $\theta \in \mathbb{R}^E$ and any finite index set N, the elements of

$$\Phi := \left\{ \phi \in \mathbb{R}^{E \times N} \;\middle|\; \forall e \in E \quad \sum_{n \in N} \phi_e^n = \theta_e \right\} \tag{7}$$

are called *reparameterizations*. For every $n \in N$,

$$\min_{x^n \in M^{**}(G)} \sum_{e \in E} \phi_e^n x_e^n \tag{8}$$

is called a *subproblem*.

For any reparametrization $\phi \in \Phi$, we have

$$D^* = \min_{x \in M^{**}(G)} \sum_{e \in E} \left(\sum_{n \in N} \phi_e^n \right) x_e \tag{9}$$

$$= \min_{x \in M^{**}(G)} \sum_{n \in N} \min_{\substack{x^n \in M^{**}(G) \\ x^n = x}} \sum_{e \in E} \phi_e^n x_e^n \quad . \tag{10}$$

Here, $x^n \in M^{**}(G)$ is a solution of subproblem n. The first equality holds by definition of Φ. The second equality is a mere reformulation of the original optimization problem in terms of a bilevel optimization problem.

Any reparameterization $\phi \in \Phi$ affords a lower bound

$$D(\phi) \;:=\; \sum_{n \in N} \min_{x^n \in M^{**}(G)} \sum_{e \in E} \phi_e^n x_e^n \;\leq\; D^* \quad . \tag{11}$$

We are interested in a reparameterization $\phi \in \Phi$ that maximizes this lower bound. However, solving

$$\max_{\phi \in \Phi} \; D(\phi) \tag{12}$$

is NP-hard because it involves solving NP-hard subproblems. Therefore, we proceed as follows. In Section 5.1, we define a constrained set of reparameterization for which we show that all subproblems can be solved efficiently. In Section 6, we define an algorithm for optimizing the reparameterization by means of subgradients and establish, for this algorithm, that it converges (in polynomial time, up to a fixed precision) to the solution D^* of the trivial LP relaxation (4) of the multicut problem. Thus, we arrive at the same bound, despite constraining the set of reparameterizations.

5.1 Constrained Reparameterization

We now define a more specific decomposition consisting of one subproblem for every edge that has a negative potential. Moreover, we constrain the set Φ of reparameterizations such that (i) each subproblem contains precisely one edge with negative potential and (ii) if, for any edge, the potential in one subproblem is negative, its potential in all other subproblems is zero. More specifically:

Definition 2: For any graph $G = (V, E)$, any potentials $\theta \in \mathbb{R}^E$ and $N := \{e \in E \mid \theta_e < 0\}$, the elements of

$$\Psi := \left\{ \psi \in \Phi \;\middle|\; \begin{array}{ll} \forall e \in N & \psi_e^e = \theta_n \\ \forall e, n \in N \mid e \neq n & \psi_e^n = 0 \\ \forall e \in E \setminus N & \psi_e^n \geq 0 \end{array} \right\} \tag{13}$$

are called *constrained reparameterizations*. For every $n \in N$,

$$\min_{x^n \in M^{**}(G^n)} \sum_{e \in E} \psi_e^n x_e^n \tag{14}$$

is called a *constrained subproblem*.

As we constrain the set of reparameterizations, clearly

$$D^{**} \ := \ \max_{\psi \in \Psi} D(\psi) \ \leq \ \max_{\phi \in \Phi} D(\phi) \ \leq \ D^* \ . \tag{15}$$

However, we show in the following that this bound is tight:

Lemma 2. *For any weighted graph $G = (V, E, \theta)$ and any constrained reparameterization $\psi \in \Psi$, $D^{**} = D^*$.*

Proof. The LP relaxation over M^{**} is exact for problems with one negative-potential edge because st-cuts correspond to a basic pairwise LP relaxation (14; 18). Moreover, each sub-problem enforces all cycle inequalities in which the single edge with negative potential in the subproblem is the pivot edge, and no other cycle inequalities. Finally, the union of all constraints enforced in any subproblem in a dual decomposition results in a lower bound whose maximum value is equal to the corresponding LP relaxation over those constraints (19).

We now show that subproblems can be solved efficiently:

Lemma 3. *For any constrained reparameterization $\psi \in \Psi$, each subproblem*

$$\min_{x^n \in M^{**}(G)} \sum_{e \in E} \psi_e^n x_e^n \tag{16}$$

can be reduced, in linear time and space, to the minimum st-cut problem.

See Appendix for the proof.

6 Bound Maximization along Subgradients

The maximization of $D(\psi)$ over all $\psi \in \Psi$ is a continuous problem with a concave, non-smooth objective function. We solve this problem by means of a projected subgradient method (15), which requires two basic steps. First is the calculation of a subgradient and second is the projection onto the feasible set.

The subgradient of $D(\psi)$ is written below.

$$(\nabla D)^n \in \arg\min_x \sum_{e \in E} \psi_e^n x_e \tag{17}$$

The projection is defined as the map $[\cdot]_\Psi : \mathbb{R}^{E \times N} \to \Psi$ such that, for all $\xi \in \mathbb{R}^{E \times N}$, all $e \in E$ and all $n \in N$:

$$[\xi_e^n]_\Psi \ := \ \begin{cases} \theta_e & \text{if } n = e, \theta_e < 0 \\ 0 & \text{if } n \neq e, \theta_e < 0 \\ \frac{[\xi_e^n]_\Phi}{\sum_{n' \in N} [\xi_e^{n'}]_\Phi} \theta_e & \text{else} \end{cases} \tag{18}$$

with the map $[\cdot]_\Phi : \mathbb{R}^{E \times N} \to \Phi$ such that, for all $\xi \in \mathbb{R}^{E \times N}$, all $e \in E$ and all $n \in N$:

$$[\xi_e^n]_\Phi \ := \ \xi_e^n + \frac{1}{N} \left(\theta_e - \sum_{n' \in N} \xi_e^{n'} \right) \tag{19}$$

Lemma 4. *For any graph $G = (V, E)$, any potentials $\theta \in \mathbb{R}^E$ and any $\xi \in \mathbb{R}^{E \times N}$, $D([\xi]_\Psi) \leq D(\xi)$.*

See Appendix for the proof.

For the subgradient update, we use a decaying step size $\lambda_k = \frac{\bar{\lambda}}{k}$ where k is the iteration number and $\bar{\lambda}$ is the initial step size, a parameter of the algorithm. Overall, we employ Algorithm 1.

Algorithm 1 Lower Bound Maximization (Subgradient)

```
Decompose: $\psi \leftarrow \theta$
for $k = 1, \ldots, k_{\max}$ do
    Get Subgradient: $(\nabla D)^n \in \arg\min_x \sum_{e \in E} \psi_e^n \, x_e$
    Get Lower Bound: $B \leftarrow \sum_n \sum_{e \in E} \psi_e^n \, (\nabla D)_e^n$
    Get Upper Bound: $V \leftarrow$ See Section ??
    Update: $\psi \leftarrow [\psi + \lambda_k \cdot \nabla D]_\Psi$
    if $V - B < \epsilon$ then
        break
    end if
end for
```

We provide an illustration of sub-gradient optimization in the appendix.

As an alternative, the dual bound can also be maximized using message passing, as outlined in the appendix.

6.1 Decoding heuristics

If we take the dual of our lower bound we recover the a primal linear program optimization that constructs the optimal partition over the basis of optimal solutions to the sub-problems. This derivation is completed in great detail in the appendix . This is however computationally very challenging to solve. We now consider four practical methods (2 of which are in the appendix) that allow going from the fractional solution resulting from the dual decomposition LP relaxation to an integer solution representing a valid graph segmentation. The first three rely on a union operation on two partitions which we define below

$$\{x^\alpha \leftarrow x^\alpha \cup x^n\} : x_e^\alpha \leftarrow \max\left[x_e^\alpha, x_e^n\right] \;\; \forall e \tag{20}$$

Full Construction: An easy mechanism to avoid solving an LP is to take the union of all partitions in Ω as x^α. This sets x^α equal to $\{x^0 \cup x^1 ... \cup x^n ...\}$.

Iterative Construction: An improvement over full construction is to construct x^α iteratively: each negative edge n is selected according to a random order and if $x_n^\alpha = 0$ then $x^\alpha \leftarrow x^\alpha \cup x^n$. Note that choosing different random orderings may produce different partitions x^α. At all stages of this procedure we retain the lowest energy partition that we have generated so far in addition to the current partition. In our experiments we run this algorithm ten times after each subgradient step over all values in ψ. Note that we apply this algorithm even before we have converged to the optimal value of ψ. Since all negative edges are cut in the solution produced by iterative construction that are cut in the solution produced by full construction but only a subset of the corresponding positive edges are cut we can guarantee that iterative construction produces a partition that does not have higher energy than the solution produced by full construction

7 Experiments

7.1 Berkeley Segmentation Data Set

We demonstrate our dual multicut algorithm on problems from the Berkeley Segmentation Data Set (BSDS) (17) and on synthetic data. The segmentation problems are defined on a super-pixel graph given by an oriented watershed transform based on the "generalized probability of boundary" (gPb) classifier (4). Each pair of adjacent super-pixels is associated with an edge whose potential equals the log odds ratio of the gPb plus an offset B:

$$\theta_e = \log\left(\frac{1 - gPb_e}{gPb_e}\right) + B \tag{21}$$

Design parameter B controls the resolution of the resulting segmentation. Extreme values make all potentials positive (negative), resulting in a single segment (one segment per supervoxel). Intermediate values yield segmentations that are among the best (1; 22) that can be achieved on this database.

We report results averaged over 200 images from the BSDS. We use the dual multicuts (DMC) as proposed here with sub-gradient optimization and the iterative construction for upper bounds. We compare with PlanarCC (22) which has been designed for and works exclusively on planar graphs. For best comparability, we rely on the same super-pixels and potentials as (22), kindly provided by the authors.

The first three plots in Fig. 2 show, for various values of B and hence for different segmentation regimes, how quickly the upper and lower bounds converge to the maximum lower bound found by any method. We plot the absolute value of this difference as a function of time, divided by the maximum lower bound found by any method. This normalization is meaningful given that the absolute energies found for different images vary widely.

While the present implementation is not as fast as the specialized PlanarCC algorithm, one part of the dual decomposition calculation is embarassingly parallel and can be distributed.

7.2 Correlation clustering in non-planar graphs

We also compared DMC to the cutting plane integer programming multicut algorithm of (1) on non-planar graphs where neither (22) nor (3) can be used. The plot shows an average over 10 instances of the following nature: each synthetic problem consists of five coupled clusters. Each cluster is a three-dimensional Cartesian lattice with a width, height and depth of six nodes. Edges inside each cluster have attractive potentials

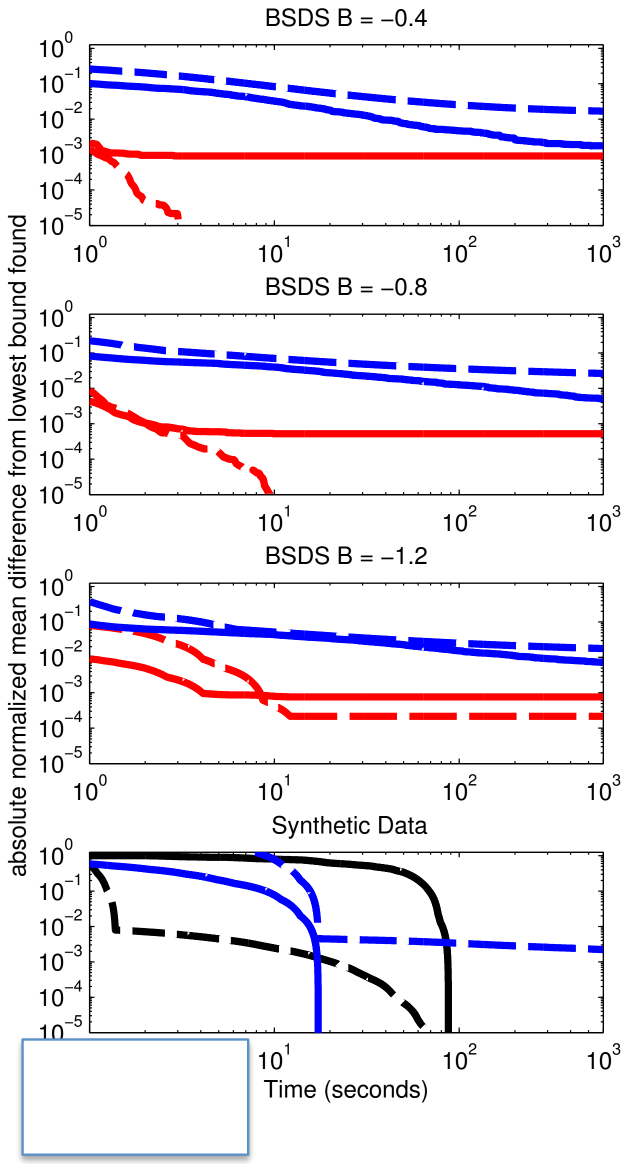


Fig. 2. Absolute normalized mean convergence of upper (solid lines) and lower (dashed lines) bounds of dual multicut (DMC, blue), PlanarCC (red) and cutting-planes ILP. For bottom plot DMC is black and ILP is blue (sorry for the confusion here this will be corrected in final version. The colors are otherwise correct)

uniformly distributed over the interval $[0, 50]$. There are five negative potential edges with value -100 connecting each pair of clusters. Each such edge connects a random node in one cluster with a random node in the other. Finally, there are ten positive potential edges with value uniformly distributed on the range $[0, 1]$ connecting each pair of clusters. Again, each such edge connects a random node in one cluster with a random node in the other.

We use iterative decoding for DMC and obtained an upper bound from the closed regions produced by multicuts at each step. On such coupled three-dimensional problems, PlanarCC is not applicable and we outperform the cutting-planes ILP approach.

8 Discussion

In this paper we present a novel approach to solving the multicut problem. We espouse the power of dual decomposition and *st*-cut solvers in a principled way to achieve the same lower bound as the basic LP relaxation.

We envisage multiple ways of speeding up the computations. Firstly, by adding in sub-problems incrementally. This takes the form of restricting most sub-problems to have zero valued potentials on all edges except for their particular negative valued edge; and removing this restriction gradually. Optimization of the lower bound follows the style of cycle pursuit (21). This has the potential to achieve dramatic speed ups and decreased memory use.

Secondly, in future work we hope to take advantage of the fact that graph cut problems produced during subgradient updates are very similar across time. Note that the n^{th} sub-problem at iteration T is similar to the n^{th} sub-problem at iteration $T + 1$ for all n and T after a few iterations of the algorithm. We can thus draw on dynamic graph cuts (13) that capitalize on previous computation for similar instances.

Thirdly, one can exploit that multiple st-cut instances can be solved independently.

Finally, we intend to apply the dual alternating direction of multipliers in order to maximize the lower bound in a way akin to message passing but avoiding the type of poor fixed points exemplified in the appendix.

Summing up, we have presented the first distributed algorithm for solving the multicut problem approximately that has guaranteed and well-understood lower and upper bounds. This may prove a good platform for future developments, some of which are outlined above.

Bibliography

[1] Andres, B., Kappes, J.H., Beier, T., Köthe, U., Hamprecht, F.A.: Probabilistic image segmentation with closedness constraints. In: ICCV (2011)

[2] Andres, B., Kröger, T., Briggman, K.L., Denk, W., Korogod, N., Knott, G., Kothe, U., Hamprecht, F.A.: Globally optimal closed-surface segmentation for connectomics. In: ECCV (2012)

[3] Andres, B., Yarkony, J., Manjunath, B.S., Kirchhoff, S., Turetken, E., Fowlkes, C.C., Pfister., H.: Segmenting planar superpixel adjacency graphs w.r.t. non-planar superpixel affinity graphs. In: EMMCVPR (2013)

[4] Arbelaez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmentation. TPAMI 33(5), 898–916 (May 2011)

[5] Bansal, N., Blum, A., Chawla, S.: Correlation clustering. In: Machine Learning. pp. 238–247 (2002)

[6] Barahona, F., Mahjoub, A.: On the cut polytope. Mathematical Programming 36(2), 157–173 (1986)

[7] Chopra, S., Rao, M.R.: The partition problem. Math. Program. 59, 87–115 (1993)

[8] Demaine, E., Emanuel, D., Fiat, A., Immorlica, N.: Correlation clustering in general weighted graphs. Theoretical Computer Science 361(2–3), 172–187 (2006)

[9] Deza, M.M., Grotschel, M., Laurent, M.: Complete descriptions of small multicut polytopes. In: Gritzmann, P., Sturmfels, B. (eds.) Applied Geometry and Discrete Mathematics, The Victor Klee Festschrift, vol. 4 (1991)

[10] Deza, M.M., Laurent, M.: Geometry of Cuts and Metrics. Springer (1997)

[11] Franc, V., Sonnenburg, S., Werner, T.: Cutting-Plane Methods in Machine Learning, chap. 7, pp. 185–218. The MIT Press, Cambridge,USA (2012)

[12] Kim, S., Nowozin, S., Kohli, P., Yoo, C.D.: Higher-order correlation clustering for image segmentation. In: NIPS (2011)

[13] Kohli, P., Torr, P.H.S.: Dynamic graph cuts for efficient inference in markov random fields. TPAMI 29(12), 2079–2088 (2007)

[14] Kolmogorov, V., Wainwright, M.J.: On the optimality of tree-reweighted max-product message-passing. CoRR abs/1207.1395 (2005)

[15] Komodakis, N., Paragios, N., Tziritas, G.: MRF energy minimization and beyond via dual decomposition. TPAMI 33(3), 531–552 (2011)

[16] Laurent, M.: A comparison of the Sherali-Adams, Lovasz-Schrijver and Lasserre relaxations for 0-1 programming. Mathematics of Operations Research 28, 470–496 (2001)

[17] Martin, D., Fowlkes, C.C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: ICCV. pp. 416–423 (2001)

[18] Rother, C., Kolmogorov, V., Lempitsky, V., Szummer, M.: Optimizing binary MRFs via extended roof duality. In: CVPR. pp. 1–8 (june 2007)

[19] Sontag, D., Globerson, A., Jaakola, T.: Introduction to dual decomposition for inference (2010)

[20] Sontag, D., Jaakkola, T.: New outer bounds on the marginal polytope. In: NIPS (2007)

[21] Sontag, D., Meltzer, T., Globerson, A., Jaakkola, T., Weiss, Y.: Tightening LP relaxations for MAP using message passing. In: UAI. pp. 503–510 (2008)

[22] Yarkony, J., Ihler, A., Fowlkes, C.: Fast planar correlation clustering for image segmentation. In: ECCV (2012)

Author Index